2008

Preserving Privacy and Fairness in Peer Data Management Systems

Hazen Elmeleegy

Ahmed Abusalah

Mourad Ouzzani

Ahmed K. Elmagarmid
Purdue University, ake@cs.purdue.edu

Report Number:
08-025

Preserving Privacy and Fairness in Peer Data Management Systems

Hazem Elmeleegy
Ahmed Abusalah
Mourad Ouzzani
Ahmed Elmagarmid

CSD TR #08-025
October 2008

Preserving Privacy and Fairness in Peer Data Management Systems

Hazem Elmeleegy[#], Ahmed Abusalah[#], Mourad Ouzzani[*], Ahmed Elmagarmid[#*]

[#]*Department of Computer Science, Purdue University*
[*]*Cyber Center, Purdue University*
West Lafayette, IN, USA

{hazem,aabusala,mourad,ake}@cs.purdue.edu

Abstract— **Peer Data Management Systems (PDMSs) promise to extend the classical data integration approach to the Internet scale. Unfortunately, some challenges remain before realizing this promise. One of the biggest challenges is preserving the privacy of the exchanged data while passing through several intermediate peers. Another challenge is protecting the mappings used for data translation. Achieving privacy preservation without being unfair to any of the peers is yet a third challenge. This paper presents a novel query answering protocol in PDMSs to address these challenges. The protocol employs a technique based on noise selection and insertion to protect the query results, and a commutative encryption-based technique to protect the mappings and ensure fairness among peers. An extensive security analysis of the protocol shows that it is resilient to seven possible types of attacks, assuming a malicious model. We implemented the protocol within an established PDMS: the Hyperion system. We conducted an experimental study using real data from the healthcare domain. The results show that our protocol introduces a moderate communication overhead compared to its non-privacy preserving counterpart and manages to achieve fairness among the peers.**

I. Introduction

Peer Data Management Systems (PDMSs) were introduced in recent years as an extension to the classical data integration paradigm, where a number of heterogeneous databases are accessible through a central mediator. In contrast to traditional data integration systems, the scale of a PDMS can be as large as the Internet. A key design principle is that the functions initially offered by the central mediator are performed in a decentralized fashion, so that the PDMS can scale with the large number of peer databases. Most of the work in the PDMS literature has focused on this problem; the provision of decentralized autonomous mechanisms for the peer databases to be able to share data seamlessly despite their heterogeneous schemas. Enabling this sharing involves having peers maintain mappings that capture the differences between their schemas and schemas of their "acquainted" peers in the PDMS. In this sense, peers may be needed to act as translators of queries and answers if they happen to occur on the path from a requesting client peer to some target server peer.

In this paper, we address an equally important challenge in PDMSs, namely privacy management. When two peers in a PDMS decide to share data, all their queries and related answers might need to be exposed to one or more intermediate peers acting as translators. Needless to say, in many cases this information can be highly sensitive and should not be publicly revealed. Consider the following cases:

1- In healthcare, two hospitals may need to exchange the medical records of patients for the purpose of consultancy or patient transfer.

2- In law enforcement, officers may need to access crime databases under different jurisdictions or even countries to be able to track down suspects.

3- In supply chain, a manufacturer needs to exchange parts data with new suppliers.

While there are naïve methods to preserve privacy in a PDMS, these methods are very inefficient and unfair to some of the peers. For instance, to answer a query, we can require all intermediate peers between the client and the server to ship all their mappings to the client, such that translation occurs at the client-side. Besides inefficiency, this approach is unfair to the intermediate peers, who are requested to give up all their mappings for no return. On the other extreme, the intermediate peers may charge the client for all the mappings they provide. However, this is unfair to the client, who only needs a very small subset of these mappings to translate the query and the result. Thus, there is a need to maintain fairness among peers. Peers acting as translators should not unnecessarily *overcharge* clients, and similarly clients should not be able to *underpay* translators.

Our proposed protocol achieves the privacy and fairness goals, while maintaining a moderate overhead, in the following way. When a server answers a query, the real result values are mixed with noise values before they are exposed to the translators. The percentage of noise depends on the privacy requirements set by the client. The noise values are selected such that a malicious translator cannot filter them out. The protocol also employs a technique based on commutative encryption, which ensures that clients only learn (and pay for) the value mappings needed to translate the query result, without getting the result's real values exposed to the translators. Although our protocol preserves privacy in addition to maintaining fairness, for simplicity, we will just refer to it as a privacy-preserving protocol. We summarize our contributions in this paper as follows:

1- We propose a privacy-preserving query answering protocol based on noise insertion and commutative encryption methods. It preserves the privacy of the query results and mappings, while maintaining fairness.

2- We formally show that the protocol satisfies the privacy requirements if a semi-honest model was assumed. We also discuss the protocol's resilience to seven possible types of attacks if a malicious model was assumed.

3- We implemented the protocol on top of an established PDMS, namely Hyperion.

4- We conducted an extensive experimental study using real data sets from a healthcare scenario. The results show that our protocol introduces a moderate processing and communication overhead, which is outweighed by the high levels of privacy and fairness it offers.

The rest of the paper is organized as follows. Section II gives an overview of the Hyperion system, which our protocol is based upon. Section III first discusses the privacy requirements for the query results and mappings in addition to the fairness considerations, and then presents the details of the privacy-preserving protocol. Section IV explains the noise selection technique. In Section V, we perform security and cost analyses of the protocol, while in Section VI we report the results of our experimental study. We discuss related work in Section VII. Finally, Section VIII concludes the paper.

II. System Overview

We consider a PDMS, in which each peer can play three different roles: a *client*, a *server*, and a *translator*. A peer is considered a client if it issues queries. It is considered a server if it processes queries and returns results from its local database. Finally, it is considered a translator if it maintains schema and data mappings with other peers, and is capable of translating data and queries to and from these peers. Since the work presented in this paper builds upon the Hyperion system, we will briefly describe the architecture of a Hyperion system and the different types of queries the system can answer.

A. *System Architecture*

Figure 1 (adapted from [17]) shows the Hyperion peer architecture. Our privacy-preserving protocol was implemented within what is called the privacy-preserving query service. For a detailed discussion about the architecture of a Hyperion peer, please refer to [10,17].

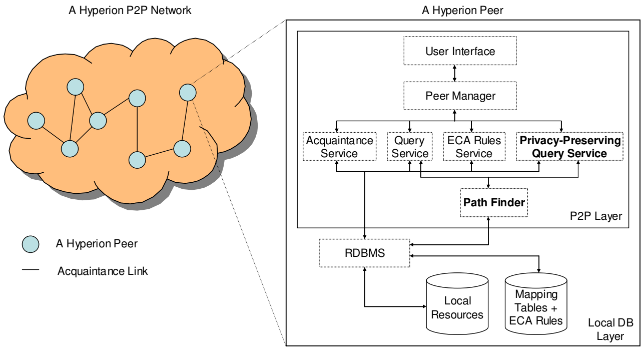


Figure 1. Hyperion Architecture

In Hyperion, as in other similar PDMSs like Piazza [20], the complete network graph is available and accessible to all the peers. Hyperion stores such graph locally in each peer. A node in the graph represents a peer and an edge represents an acquaintance link between two peers, which implies that mapping tables are available to translate data and queries between these two peers. Weights can be assigned to edges to reflect one or several factors about the acquaintance link such as reliability, latency, usage cost, or a combination of these factors. The mapping tables are typically replicated in both acquainted peers. Thus, any two peers having a common acquaintance (common neighboring peer) can communicate directly using the schema of their common acquaintance. It follows that the original graph can be augmented by adding *shortcut edges* between any two nodes that are two links apart. Figure 2 shows how the example graph shown in Figure 1 is augmented. The weight of a shortcut edge can either be dependant on the weights of its two "shortcutted" edges or not (e.g. the sum of usage costs, the product of reliability values, or just the latency between the peers at both ends of the shortcut edge).

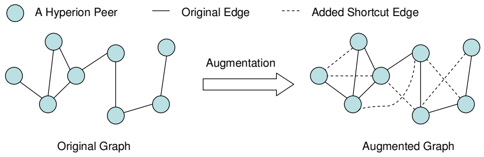


Figure 2. Graph Augmentation

B. *Query Answering*

There are two types of queries that can be answered in a PDMS: *broadcast* queries and *targeted* queries. For broadcast queries, the client sends the query to its acquaintances which in turn forward it to their own acquaintances and so on, thus propagating the query to the whole network. Before a peer forwards the query, it is translated first to conform to the schema of the following peer. Each peer receiving the query and having a local database executes it locally. Finally, all the peers that could find relevant local results send those results back to the client; each through the same path that was used to deliver the query to it, but in the reverse direction. The results get translated by the intermediate peers on their way back to the client. For targeted queries, the client specifies a *target* peer from where to get the results. The shortest path to the target peer is first computed on the augmented graph. The query is then sent (and translated) along the discovered path, and the corresponding result is also sent (and translated) along the same path, but in the reverse direction. For the rest of the discussion, we will focus on targeted queries rather than broadcast queries. A broadcast query can be viewed as a collection of targeted queries in terms of delivering results from different servers back to the client. Each result will travel back through a certain path, similar to targeted queries.

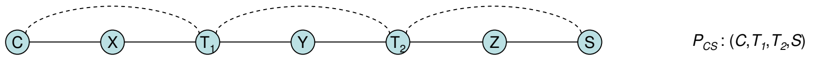
In Hyperion, mappings used for query translation depend on mapping tables, which are value-to-value mappings. More generally, in other PDMSs and data integration systems, mappings can either be Local-As-View (LAV), Global-As-View (GAV), or Local-and-Global-As-View (GLAV). We emphasize that our work is independent from the way queries are translated. It is also worth mentioning that the version of Hyperion we used is an extension to the one presented in [17]. We have added support for targeted queries as well as result translation on the path back to the client.

III. Privacy and Fairness in Query Answering

A. *Privacy and Fairness Requirements*

1) Privacy of the Query Result: The result R of a query Q consists of l columns ($R_1, R_2,\ldots, R_l$) representing the attributes ($a_1, a_2,\ldots, a_l$). D_j denotes the domain of attribute a_j, and UR_j denotes the unique values in R_j, $j\in[1,l]$. If Q, R, R_j, UR_j, a_j, and D_j follow the schema of some peer X, then we refer to them as Q^X, R^X, R_j^X, UR_j^X, a_j^X, and D_j^X respectively, $j\in[1,l]$.

The main role of any PDMS's query answering protocol can be stated as follows. Consider a client peer C issuing a query Q^C to be executed on a target server peer S, such that the path



(a) Part of the peer network and path from client to server

CPatients

CPID
p_{10}
p_{15}
p_{22}
p_{35}
p_{37}

CMedications

CMID
m_1
m_2
m_3
m_4
m_5
m_6
m_7
m_8
m_9
m_{10}

CTreatments

CPID	*CMID*
p_{15}	m_1
p_{15}	m_6
p_{37}	m_4

CPID_2_XPID

CPID	*XPID*
$\mathcal{X}$	$\mathcal{X}$

CMID_2_XMID

CMID	*XMID*
m_1	$m_{1'}$
m_2	$m_{2'}$
m_3	$m_{3'}$
m_4	$m_{4'}$
m_5	$m_{5'}$
m_6	$m_{6'}$
m_7	$m_{7'}$
m_8	$m_{8'}$
m_9	$m_{9'}$
m_{10}	$m_{10'}$

(b) Client local tables and mapping tables

SPatients

SPID
p_5
p_{17}
p_{19}
p_{35}
p_{51}

SMedications

SMID
m_a
m_b
m_c
m_d
m_e
m_f
m_g
m_h
m_i
m_j

STreatments

SPID	*SMID*
p_5	m_b
p_5	m_e
p_{17}	m_c
p_{19}	m_a
p_{19}	m_d
p_{19}	m_e
p_{35}	m_f
p_{35}	m_h
p_{51}	m_c
p_{51}	m_i
p_{51}	m_j

ZPID_2_SPID

ZPID	*SPID*
$\mathcal{X}$	$\mathcal{X}$

ZMID_2_SMID

ZMID	*SMID*
$m_{a'}$	m_a
$m_{b'}$	m_b
$m_{c'}$	m_c
$m_{d'}$	m_d
$m_{e'}$	m_e
$m_{f'}$	m_f
$m_{g'}$	m_g
$m_{h'}$	m_h
$m_{i'}$	m_i
$m_{j'}$	m_j

(c) Server local tables and mapping tables

XPID_2_T1PID

XPID	*T1PID*
$\mathcal{X}$	$\mathcal{X}$

XMID_2_T1MID

XMID	*T1MID*
$m_{1'}$	$m_{1''}$
$m_{2'}$	$m_{2''}$
$m_{3'}$	$m_{3''}$
$m_{4'}$	$m_{4''}$
$m_{5'}$	$m_{5''}$
$m_{6'}$	$m_{6''}$
$m_{7'}$	$m_{7''}$
$m_{8'}$	$m_{8''}$
$m_{9'}$	$m_{9''}$
$m_{10'}$	$m_{10''}$

T1PID_2_YPID

T1PID	*YPID*
$\mathcal{X}$	$\mathcal{X}$

T1MID_2_YMID

T1MID	*YMID*
$m_{1''}$	m_α
$m_{2''}$	m_β
$m_{3''}$	m_χ
$m_{4''}$	m_δ
$m_{5'}$’	m_ε
$m_{6''}$	m_ϕ
$m_{7''}$	m_γ
$m_{8''}$	m_η
$m_{9''}$	m_ι
$m_{10''}$	m_φ

(d) Translator T_1 mapping tables

YPID_2_T2PID

YPID	*T2PID*
$\mathcal{X}$	$\mathcal{X}$

YMID_2_T2MID

YMID	*T2MID*
m_α	$m_{a''}$
m_β	$m_{b''}$
m_χ	$m_{c''}$
m_δ	$m_{d''}$
m_ε	$m_{e''}$
m_ϕ	$m_{f''}$
m_γ	$m_{g''}$
m_η	$m_{h''}$
m_ι	$m_{i''}$
m_φ	$m_{j''}$

T2PID_2_ZPID

T2PID	*YPID*
$\mathcal{X}$	$\mathcal{X}$

T2MID_2_ZMID

T2MID	*ZMID*
$m_{a''}$	$m_{a'}$
$m_{b''}$	$m_{b'}$
$m_{c''}$	$m_{c'}$
$m_{d''}$	$m_{d'}$
$m_{e''}$	$m_{e'}$
$m_{f''}$	$m_{f'}$
$m_{g''}$	$m_{g'}$
$m_{h''}$	$m_{h'}$
$m_{i''}$	$m_{i'}$
$m_{j''}$	$m_{j'}$

(e) Translator T_2 mapping tables

Figure 3. Peer network and peer tables in Example 3.1

P_{CS} from C to S is $(C,T_1,T_2,\ldots,T_n,S)$, where $(T_1,T_2,\ldots,T_n)$ are n intermediate translator peers. A PDMS query answering protocol $\wp$ should be able to translate Q^C into Q^S, execute Q^S on S to obtain R^S, translate R^S to R^C, and finally return R^C to C.

The knowledge leaked through $\wp$ to any peer T_h during the execution of $\wp$ is denoted by $\Re(\wp,T_h)$, $h\in[1,n]$. In what follows, we introduce the notion of *K-protection*, which allows the client to specify the privacy requirements for the query result.

Definition 3.1 *K*-protection *Given a vector* $K=(k_1,k_2,\ldots,k_l)$, *where* k_j *is an integer, a protocol* $\wp$ *for returning* R^C *given* Q^C, *S and* P_{CS} *is said to provide K-protection to* R^C *if* $Pr(v\in R_j^X|\Re(\wp,X))\leq max(1/k_j,|UR_j^S|/|D_j^S|)$, $\forall\ v\in D_j^X$, *where X is neither C nor S,* $j\in[1,l]$.

The above definition states that for a protocol $\wp$ to satisfy the *K*-protection requirement, whatever information it passes to a peer *X* (other than *C* and *S*) *should not increase* the certainty of *X* about the result beyond a certain limit. In particular, *X* should not be able to determine, with a probability above a certain threshold, that any given value *v* belongs to R^X (the translation of R^C to *X*'s schema). This threshold is determined by the maximum of two values: (1) a user-specified value; and (2) the probability that *v* belongs to R^X knowing the number of values in *v*'s domain and the number of unique values in the result from that domain (since these numbers are not considered sensitive under the *K*-protection model). The following example illustrates how to check if a protocol satisfies the *K*-protection requirement. We will also be referring to it throughout the discussion.

Example 3.1

We consider a scenario from the healthcare domain, where peers represent hospital databases storing information about their patients and the medications they were given. Figure 3a depicts part of the P2P network, including a client peer C, a server peer S, and the path P_{CS} *between them* (C,T_1,T_2,S). *Along* P_{CS}, *two intermediate translator peers exist:* T_1 *and* T_2. *Data and queries exchanged between C and* T_1 *conform to the schema of their common acquaintance, X. Similarly, data and queries exchanged between* T_1 *and* T_2, *and between* T_2 *and S conform to Y's and Z's schemas respectively.*

Figures 3b and 3c show the tables in the databases of C and S, while Figures 3d and 3e show the mapping tables of both translators. Note that the mapping tables for patient IDs map every value to itself as denoted by the variable $\mathcal{X}$*. We assume that all peers use similar patient IDs (e.g. SSN). The schemas of all peers are shown to be similar for ease of exposition.*

C issues a query Q^C *to request the medication information stored in S for patient* p_{35}. Q^C *is shown in Figure 4a*. *The* `PATH` *clause specifies* P_{CS}, *while the* `K-PROTECTION` *clause specifies the K vector, which was specified as (2), i.e.,* $k_1=2$.

Q^S *(Figure 4b) represents the query* Q^C *after being translated to follow S's schema.* R^S, *the result corresponding* *to* Q^S *is shown in Figure 4c, and finally the translation of* R^S *back to C's schema,* R^C *is shown in Figure 4d.*

If T_2 *knows that the number of unique values of SMID in* R^S *is 2 (*$|UR_1^S|=2$*), and knows that the domain of SMID in S has 10 unique values (*$|D_1^S|=10$*), then given no additional information,* T_2 *can only conclude that* $Pr(v \in R_1^{T_2})=2/10=0.2$*, for every* $v \in D_1^{T_2}$*. The same applies to* T_1*. Thus, with* k_1 *specified as 2, the threshold value is given by max (1/2,2/10)=0.5.*

```
Q^C: SELECT CMID                  Q^S: SELECT SMID
     FROM CTreatments                  FROM STreatments
     WHERE CPID = 'P35'                WHERE SPID = 'P35'
     PATH C,T1,T2,S                    PATH C,T1,T2,S
     K-PROTECTION CMID 2               K-PROTECTION SMID 2
```

(a) Query Q^C on client

(b) Query Q^S on server

R^S

SMID
m_f
m_h

(c) Query result R^S on server

R^C

CMID
m_6
m_8

(d) Query result R^C on client

Figure 4. Queries and query results in Example 3.1

To check if a certain protocol provides K-protection to R^C*, consider the two protocols* $\wp_1$ *and* $\wp_2$*.* $\wp_1$ *represents the standard non-privacy-preserving protocol, where the result has to pass through* T_1 *and* T_2*. With* $\wp_1$*,* T_2 *learns precisely the two values in* R^{T_2}*, i.e.* $Pr(m_{f'} \in R_1^{T_2} | \mathfrak{R}(\wp_1,T_2))=1$*, which is larger than the threshold. Clearly,* $\wp_1$ *does not preserve the privacy of* R^C *based on the K-protection model.*

$\wp_2$ *requires* T_1 *and* T_2 *to send all their mapping tables to S where all the result translation occurs from* R^S *all the way to* R^X*, which is finally sent to C to be translated locally to* R^C*. With* $\wp_2$*, neither* T_1 *nor* T_2 *learns any additional information about* R^{T_1} *or* R^{T_2} *respectively. Thus,* $Pr(v_1 \in R_1^{T_1} | \mathfrak{R}(\wp_2,T_1))=Pr(v_2 \in R_1^{T_2} | \mathfrak{R}(\wp_2,T_2))=0.2$*, for every* $v_1 \in D_1^{T_1}$ and $v_2 \in D_1^{T_2}$ *(using the knowledge about unique values of SMID and its domain). This value is below the threshold. Thus,* $\wp_2$ *preserves* R^C*'s privacy based on the K-protection model. However,* $\wp_2$ *is very inefficient since the mapping tables of* T_1 *and* T_2 *may be quite large. Also,* $\wp_2$ *violates the mapping tables' privacy, as will be shown next.* □

2) Privacy of the Mapping Tables and Fairness Considerations: Usually, peers in a PDMS will act as translators for other peers based on some incentives. For instance, peers can provide their translation services for a service fee that is based on the number of mapping entries requested by the client. One issue with this model is that after purchasing a mapping entry from some translator, a client can then distribute that entry to other peers. These peers will not have to pay any fees to the original owner of the entry. This is considered unfair to the translator. Conversely, if a translator forces a client to purchase more entries than what the client actually needs, the situation is reversed, and it becomes unfair to the client. Therefore, it is desirable to achieve fairness between clients and translators, especially when some peers mostly act as clients, while others mostly act as translators.

We will define the fairness of a protocol with respect to translators and clients. We model the protocol's fairness based on the total revenue realized by every translator in a PDMS.

Definition 3.2 protocol fairness. *A protocol is considered to be fair if the revenue of any translator T at any point in time* $t=\tau$ *is bounded by:*

a) (Upper bound) the sum of costs of every request made by every client before $t=\tau$ *to obtain one of T's mapping entries.*

b) (Lower bound) the sum of costs of every "unique" request made by every client before $t=\tau$ *to obtain one of T's mapping entries. Uniqueness is considered independently for every client.*

The rationale behind the above definition is as follows. On one hand, if *T*'s revenue is less than the specified lower bound, then some clients have obtained some of *T*'s mapping entries to translate their query results from sources other than *T*, for example through some sort of a "black market". This is clearly unfair to *T*. On the other hand, if *T*'s revenue is greater than the specified upper bound, then at least one client *C* has paid *T* more than the total cost of all its requested mapping entries. This time it is unfair to *C*.

In other words, according to this definition of fairness, a client is allowed to cache some of the mapping entries it has previously purchased from a translator *T*, and then re-use them in translating newer query results. However, neither should the client get involved in any sort of black market activity, nor should the translator charge clients more than the cost of what they actually need. A stricter definition of fairness may be also considered, such as a definition that does not allow clients to cache and re-use any of their purchased mapping entries. However, for the rest of the discussion, we will only consider Definition 3.2 for protocol fairness.

B. Privacy Preserving Protocol

Our proposed privacy preserving protocol relies on two key ideas. The first idea is to insert selected noise values in the query result before requesting mapping entries from the translators. This is geared towards preserving the privacy of the query result. The second idea is concerned with preserving the privacy of the mapping tables while maximizing fairness between clients and translators. In this case, we use a technique based on commutative encryption [16] to ensure that the client only knows the mapping entries necessary to translate the result of its query, while the translator still cannot distinguish between the real values that actually exist in the query result and the noise values inserted into it.

Before discussing the details of the protocol, we recall that the input of the protocol is a query Q^C that conforms to *C*'s schema, a path P_{CS} $(C,T_1,T_2,\ldots,T_n,S)$ that connects *C* to *S*, and the *K* vector, which defines the query result privacy requirements; while the output is the query result R^C which is the translation of R^S. For ease of exposition, we assign the alias names T_0 and $T_{(n+1)}$ to the client and the server in addition to their original names *C* and *S* respectively. We will use these aliases interchangeably with the original names. We assume that data and queries sent between peers T_h and $T_{(h+1)}$ follow the schema of some peer T'_h, $h \in [0,n+1]$. T'_h can either be T_h, $T_{(h+1)}$, or a common acquaintance of both T_h and $T_{(h+1)}$ in case they are not directly connected.

The protocol has three phases: the *query delivery* phase (phase I), *the result and mappings collection* phase (phase II), and the *mappings decryption and result translation* phase (phase III). The protocol uses message passing between the participating peers, where six types of messages are involved. Table 1 lists the six message types along with the senders/receivers for each type, and the protocol phase in which messages from each type are used. In general, before peers communicate, they first authenticate each other to guard against impersonation attacks. We now discuss how the protocol operates in the three phases.

TABLE 1. PROTOCOL MESSAGE TYPES

Message Type	Phase	Sender(s)	Receiver(s)
Query_Msg (QM)	I	Client and Translators	Server and Translators
Mapping_Req_Msg (MRqM)	II	Server and Translators	Translators
Mapping_Resp_Msg (MRsM)	II	Translators	Client
Result_Msg (RM)	II	Server	Client
Decryption_Req_Msg (DRqM)	III	Client	Translators
Decryption_Resp_Msg (DRsM)	III	Translators	Client

Phase I: Query Delivery

The main goal of this phase is to deliver the client's query to the target server peer after translating it to a form that conforms to the server's schema. Since, based on our problem definition, this phase does not involve privacy preservation; the protocol operates in the same way as the non-privacy-preserving protocol. The Query_Msg is forwarded from C to S passing through T_1 to T_n. Lines 1.1-1.3 in Algorithm 1 show how translator peers handle an incoming Query_Msg. Later, in Section V, we will show that a modification needs to be applied to this phase to guard against a certain type of attacks.

Phase II: Result and Mappings Collection

This phase begins when S receives a Query_Message. The way S handles the Query_Message is also outlined in Algorithm 1 (lines 1.4-1.16). S first translates the incoming query to get Q^S, which is executed to obtain the result R^S (lines 1.5,1.6). Then, the unique values in R^S are extracted to form UR^S (line 1.7). To ensure K-protection of the result, for every unique value in R_j^S, a list of k_j-1 noise values are selected from D_j^S (the domain of a_j^S). The noise values are then merged with the values of UR_j^S to form NUR_j^S (lines 1.8-1.9). The technique used to select the noise values is explained in details in Section IV. The values in NUR_j^S are shuffled (line 1.10) to make sure that real values cannot be isolated from noise values. After NUR^S is created, S translates R^S and NUR^S to $R^{T'_n}$ and $NUR^{T'_n}$ respectively, where the former is sent to C in a Result_Msg, while the latter is sent to T_n in a Mapping_Req_Msg (lines 1.11-1.14).

The way T_n (as well as any other translator) handles the Mapping_Req_Message is shown in Algorithm 2. T_n creates a mapping from the received values of $NUR^{T'_n}$ to their corresponding values in $T'_{(n-1)}$ (lines 2.1,2.2). However, the values in $T'_{(n-1)}$ are encrypted using the key of T_n. We refer to such mapping from the plaintext values in T'_n to the encrypted values in $T'_{(n-1)}$ by $E_n(M_n)$. When the mapping is from the plaintext values in T'_n to the plaintext values in $T'_{(n-1)}$, we simply refer to it as M_n. To construct $E_n(M_n)$ efficiently, T_n keeps the values in its mapping tables both in plaintext and encrypted formats. This way, it does not have to perform encryption each time it receives a Mapping_Req_Msg. After constructing $E_n(M_n)$, T_n sends it to C in a Mapping_Resp_Msg (line 2.3). T_n also translates $NUR^{T'_n}$ to $NUR^{T'_{(n-1)}}$, shuffles $NUR^{T'_{(n-1)}}$, and sends it in a new Mapping_Req_Msg to the preceding translator on the path, $T_{(n-1)}$ (lines 2.5-2.7). Shuffling guarantees that $NUR^{T'_{(n-1)}}$ cannot be correlated with $NUR^{T'_n}$, which can result in the decryption of $E_n(M_n)$. This is important to prevent certain attacks, as will be discussed in Subsection V-A. On receiving the new Mapping_Req_Message, $T_{(n-1)}$ handles it in a similar way, and Mapping_Req_Msg's keep on propagating backwards until T_1 is reached (line 2.4).

When C receives a Result_Msg from S, or a Mapping_Resp_Msg from T_h, $h \in [1,n]$, it stores the incoming result and mappings locally, until all of them have arrived. At this point, phase II ends. The handling of both messages is described in Algorithms 3 and 4 respectively.

Phase III: Mappings Decryption and Result Translation

This phase begins when C receives the result ($R^{T'_n}$) from S and the mappings ($E_h(M_h)$, $h \in [1,n]$) from all the translators. Lines 3.2-3.6 in Algorithm 3 and lines 4.2-4.6 in Algorithm 4 are similar and they both describe the beginning of phase III. The lines in Algorithm 3 are executed if C receives the result after it has received all the translators' mappings. Otherwise, the lines in Algorithm 4 are executed instead.

At first, C extracts the unique values from $R^{T'_n}$ to generate $UR^{T'_n}$ and translates $UR^{T'_n}$ using $E_n(M_n)$ (lines 3.3,3.4 or 4.3,4.4). The translation gives the corresponding values in $T'_{(n-1)}$ encrypted with T_n's key. We refer to the generated values by $E_n(UR^{T'_{(n-1)}})$. C then re-encrypts each value in $E_n(UR^{T'_{(n-1)}})$ with its own key to generate $E_0(E_n(UR^{T'_{(n-1)}}))$, and sends such doubly-encrypted values to T_n in a Decryption_Req_Msg (lines 3.5,3.6 or 4.5,4.6).

T_n handles the Decryption_Req_Msg as shown in Algorithm 5. It decrypts the incoming values ($E_0(E_n(UR^{T'_{(n-1)}}))$) from C with T_n's key. Since commutative encryption is used, this step removes the prior encryption with T_n's key, thus generating $E_0(UR^{T'_{(n-1)}})$ (line 5.1). T_n then sends $E_0(UR^{T'_{(n-1)}})$ back to C in a Decryption_Resp_Msg (line 5.2).

Algorithm 6 shows the behavior of C once it receives the Decryption_Resp_Msg. C re-decrypts the incoming values ($E_0(UR^{T'_{(n-1)}})$) with its own key, which generates the plaintext values in $T'_{(n-1)}$, or $UR^{T'_{(n-1)}}$ (line 6.1). C also extracts the unique values from $R^{T'_n}$ to get $UR^{T'_n}$ (line 6.2), and creates a mapping M_n from the values of $UR^{T'_n}$ to the values of $UR^{T'_{(n-1)}}$ (line 6.3). C uses M_n to translate $R^{T'_n}$ into $R^{T'_{(n-1)}}$, and stores it in place of $R^{T'_n}$ (lines 6.4,6.5). C then repeats the same process with every translator preceding T_n until $R^{T'_0}$ is obtained (lines 6.6-6.9). At this point, C translates $R^{T'_0}$ to R^C using its own mapping tables (lines 6.11,6.12).

Algorithm 1: Handle_Query_Msg ($Q^{T'_{(h-1)}}$, P_{CS}, K)

```
      $T_h$ : the current peer
1.1-    if ($T_h$ is not $S$)
1.2-        Translate $Q^{T'_{(h-1)}}$ to $Q^{T'_h}$
1.3-        Send a new Query_Msg($Q^{T'_h}$, $P_{CS}$,$K$) to $T_{(h+1)}$ (if $h<l$) or
            to $S$ (if $h=n$)
1.4-    else  //$T_h$ is $S$; i.e. $h$-1=$n$
1.5-        Translate $Q^{T'_n}$ to $Q^S$
1.6-        Execute $Q^S$ to get $R^S$
1.7-        Extract unique values from $R^S$ to get $UR^S$
```

```
1.8-          for each attribute $a_j^S$ in $R^S$
1.9-             Construct $NUR_j^S$ as the union of $UR_j^S$ and min(($k_j$-1)×|$UR_j^S$|,|$D_j^S$|-|$UR_j^S$|) additional noise values
1.10-            Shuffle $NUR_j^S$
1.11-         Translate $NUR^S$ to $NUR^{T'_n}$
1.12-         Translate $R^S$ to $R^{T'_n}$
1.13-         Send a new Mapping_Req_Msg($NUR^{T'_n}$, $P_{CS}$) to $T_n$
1.14-         Send a new Result_Msg($R^{T'_n}$) to $C$
1.15-      return
```

Algorithm 2: Handle_Mapping_Req_Msg ($NUR^{T'_h}$, P_{CS})

```
   $T_h$ : the current peer
2.1-    Translate $NUR^{T'_h}$ using $T'_{(h-1)}$ values encrypted with the key of $T_h$ to get $E_h(NUR^{T'_{(h-1)}})$
2.2-    Join each value in $NUR^{T'_h}$ with its corresponding value in $E_h(NUR^{T'_{(h-1)}})$ to get the encrypted mapping $E_h(M_h)$
2.3-    Send a new Mapping_Resp_Msg($E_h(M_h)$) to $C$
2.4-    if ($T_{(h-1)}$ is not $C$)
2.5-       Translate $NUR^{T'_h}$ to $NUR^{T'_{(h-1)}}$
2.6-       Shuffle $NUR^{T'_{(h-1)}}$
2.7-       Send a new Mapping_Req_Msg($NUR^{T'_{(h-1)}}$, $P_{CS}$) to $T_{(h-1)}$
2.8-    return
```

Algorithm 3: Handle_Result_Msg ($R^{T'_n}$)

```
   // the current peer must be $C$
3.1-    Store $R^{T'_n}$
3.2-    if (all translators' mappings and $R^{T'_n}$ are received)
3.3-       Extract unique values from $R^{T'_n}$ to get $UR^{T'_n}$
3.4-       Translate $UR^{T'_n}$ using $E_n(M_n)$ to get $E_n(UR^{T'_{(n-1)}})$
3.5-       Encrypt $E_n(UR^{T'_{(n-1)}})$ with $C$'s key to get $E_0(E_n(UR^{T'_{(n-1)}}))$
3.6-       Send a new Decryption_Req_Msg($E_0(E_n(UR^{T'_{(n-1)}}))$) to $T_n$
3.7-    return
```

Algorithm 4: Handle_Mapping_Resp_Msg ($E_h(M_h)$)

```
   // the current peer must be $C$
4.1-    Store $E_h(M_h)$
4.2-    if (all translators' mappings and $R^{T'_n}$ are received)
4.3-       Extract unique values from $R^{T'_n}$ to get $UR^{T'_n}$
4.4-       Translate $UR^{T'_n}$ using $E_n(M_n)$ to get $E_n(UR^{T'_{(n-1)}})$
4.5-       Encrypt $E_n(UR^{T'_{(n-1)}})$ with $C$'s key to get $E_0(E_n(UR^{T'_{(n-1)}}))$
4.6-       Send a new Decryption_Req_Msg($E_0(E_n(UR^{T'_{(n-1)}}))$) to $T_n$
4.7-    return
```

Algorithm 5: Handle_Decryption_Req_Msg ($E_0(E_h(UR^{T'_{(h-1)}}))$)

```
   $T_h$ : the current peer
5.1-    Decrypt $E_0(E_h(UR^{T'_{(h-1)}}))$ with the key of $T_h$ to get $E_0(UR^{T'_{(h-1)}})$  //uses the commutative encryption property
5.2-    Send a new Decryption_Resp_Msg($E_0(UR^{T'_{(h-1)}})$) to $C$
5.3-    return
```

Algorithm 6: Handle_Decryption_Resp_Msg ($E_0(UR^{T'_{(h-1)}})$)

```
   // the current peer must be $C$
6.1-   Decrypt $E_0(UR^{T'_{(h-1)}})$ with the key of $C$ to get $UR^{T'_{(h-1)}}$
6.2-   Extract unique values from each attribute in $R^{T'_h}$ to get $UR^{T'_h}$
6.3-   Join each value of $UR^{T'_h}$ with its corresponding value from $UR^{T'_{(h-1)}}$ to get the non-encrypted mapping $M_h$
6.4-   Translate $R^{T'_h}$ with $M_h$ to get $R^{T'_{(h-1)}}$
6.5-   Store $R^{T'_{(h-1)}}$ in place of $R^{T'_h}$
6.6-   if ($h$>1)  // $R^{T_{(h-1)}}$ is not $R^C$
6.7-      Translate $UR^{T'_{(h-1)}}$ using $E_{(h-1)}(M_{(h-1)})$ to get $E_{(h-1)}(UR^{T'_{(h-2)}})$
6.8-      Encrypt $E_{(h-1)}(UR^{T'_{(h-2)}})$ with $C$'s key to get $E_0(E_{(h-1)}(UR^{T'_{(h-2)}}))$
6.9-      Send a Decryption_Req_Msg($E_0(E_{(h-1)}(UR^{T'_{(h-2)}}))$) to $T_{(h-1)}$
6.10-  else
6.11-     Translate $R^{T'_{(h-1)}}$ to $R^C$
6.12-     Report $R^C$ to user
6.13-  return
```

Example 3.2

Continuing on Example 3.1, Figure 5 shows how the query in Figure 4a is executed when the privacy-preserving protocol is used. The figure shows the contents of the 12 messages exchanged between the peers during query processing. If we refer to the privacy-preserving protocol by $\wp_3$, then the only information $\wp_3$ passes to T_2 about R^{T_2} is that it contains two values, which belong to $\{m_{c''},m_{f''},m_{h''},m_{i''}\}$. Therefore, $Pr(v \in R_1^{T_2} | \mathfrak{R}(\wp_3,T_2)) = 2/4 = 0.5$, which is equal to the threshold given by max(1/2,2/10)=0.5. Similarly, we can find that $Pr(v \in R_1^{T_2} | \mathfrak{R}(\wp_3,T_2)) \leq 0.5$, for every $v \in D_1^{T_2}$. The same applies to T_1. Thus, unlike $\wp_1$, $\wp_3$ does preserve the privacy of R^C based on the K-protection model. Moreover, unlike $\wp_2$, $\wp_3$ does not require the transfer of the complete mapping tables of T_1 and T_2. Thus, $\wp_3$ is more efficient and more fair to T_1 and T_2 than $\wp_2$. □

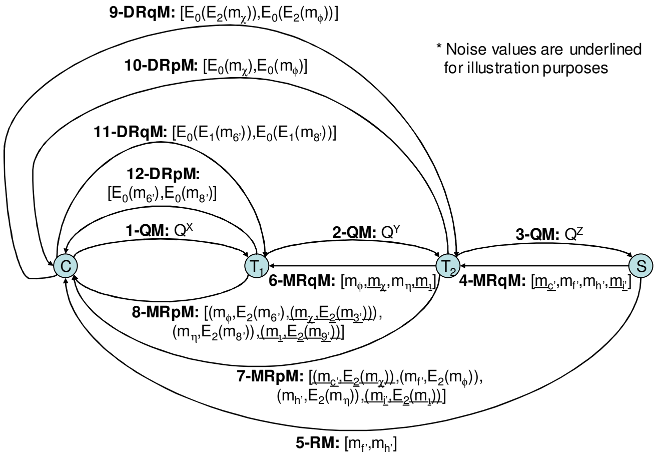

9-DRqM: [E0(E2(mχ)),E0(E2(mϕ))]
10-DRpM: [E0(mχ),E0(mϕ)]
* Noise values are underlined for illustration purposes
11-DRqM: [E0(E1(m6')),E0(E1(m8'))]
12-DRpM: [E0(m6'),E0(m8')]
1-QM: QX
2-QM: QY
3-QM: QZ
C
T1
T2
S
6-MRqM: [mϕ,mχ,mη,mι]
4-MRqM: [mc',mf',mh',mi']
8-MRpM: [(mϕ,E2(m6')),(mχ,E2(m3'))), (mη,E2(m8')),(mι,E2(m9'))]
7-MRpM: [(mc',E2(mχ)),(mf',E2(mϕ)), (mh',E2(mη)),(mi',E2(mι))]
5-RM: [mf',mh']

Figure 5. Privacy-preserving protocol operation for Example 3.2

IV. Noise Selection

Careful selection of noise values to be mixed with real values in the result is crucial to the protocol's effectiveness. For example, if the selected noise values were purely random, then they can be easily filtered out By issuing the same query multiple times and comparing multiple instances of the result, the real values will remain unchanged, while the noise values may be different, and hence they can be eliminated.

A good scheme for noise selection should guarantee that the reported result for the same query (including real and noise values) should remain the same every time the query is issued. Moreover, if the query result is changed, then the newly reported result for the query should be a *superset* of the previously reported results for the same query.

Before presenting our proposed scheme for noise selection, we first discuss two requirements concerning the peers' databases. First, database records need to be time-stamped. This is not an expensive requirement, especially with the decreasing cost of storage. Moreover, a peer whose existing database has no timestamps can still timestamp all its records with the current time before joining the PDMS. Second, the domains for the values appearing in a query result (i.e. D_j^S, $j\in[1,l]$) should not be shrunken over time. This is also an easy requirement, because even if there was a need to remove certain domain values, they can just be marked as removed instead of physically removing them.

Given the result R^S, our goal is to select (k_j-1) noise values for every unique value in R_j^S to generate NUR_j^S, $j\in[1,l]$, such that the selected noise values remain consistent every time the query is executed. The proposed scheme works as follows.

1. Extract the unique values from R_j^S to get UR_j^S, and associate each unique value with the earliest timestamp among the timestamps of all the records of R^S containing that value for R_j^S. This implies that a value would not have appeared in the query result before the time indicated by its associated timestamp.
2. Sort the values in UR_j^S based on their associated timestamps.
3. Create $|UR_j^S|$ pseudo-random number generators (PRNGs), where the seed of the i^{th} PRNG is a function of the hash value of the concatenation of the first i values in UR_j^S, $i\in[1,|UR_j^S|]$.
4. Use the i^{th} PRNG to generate the (k_j-1) noise values for the i^{th} real value in UR_j^S and insert those noise values along with the real value into NUR_j^S. This step is repeated for every $i\in[1,|UR_j^S|]$ in an ascending order of i, and is performed as follows.
   a. Let D'^S_j be D_j^S after removing all values already in NUR_j^S and all values whose timestamp in D_j^S is newer than the timestamp of the i^{th} real value in UR_j^S.
   b. Generate $|D'^S_j|$ random numbers, and associate each one of them with a value in D'^S_j.
   c. Sort D'^S_j according to the associated random numbers (to get a random permutation).
   d. Return the top (k_j-1) values.

Example 4.1

For the query Q^S (Figure 3b), the corresponding result, R^S, or similarly UR_1^S, is $\{m_f,m_h\}$ (Figure 3d). Assuming that the timestamp of m_f is earlier than that of m_h, then the scheme first inserts m_f into NUR_1^S. Then, a single noise value is selected corresponding to m_f (recall that k_1=2) using the following query (follows the MySQL syntax).

```
Q1: select SMID from SMedications
    where SMID not in ('mf')
    and TS < ts_mf
    order by rand(seed1) limit 1
```

The variable `seed1` is an integer representing the leftmost four bytes of the hash value of the string "m_f". It is passed as the seed of a PRNG used to order the `SMID` values randomly. `TS` is the timestamp column in SMedications and `ts_mf` is the timestamp of m_f. With the `TS` predicate, we ensure that the noise value corresonding to m_f only depends on SMID values which existed at a time when UR_j^S did not include m_f. This way, if `Q1` was re-issued at a later time, newly added SMID values will not change the noise selection for m_f. Assume that the random value generated by this query is m_c, then after inserting m_c into NUR_1^S, NUR_1^S now becomes $\{m_f,m_c\}$. We then insert the next value in UR_1^S (m_h) into NUR_1^S, such that NUR_1^S=$\{m_f,m_c,m_h\}$. Similar to Q_1, the following query is used to get the noise value corresponding to m_h.

```
Q2: select SMID from SMedications
    where SMID not in ('mf', 'mc', 'mh')
    and TS < ts_mh
    order by rand(seed2) limit 1
```

The variable `seed2` is computed similar to `seed1` except that the string "m_f" is replaced with "m_f:m_h". If the above query returns the value m_i, then NUR_1^S becomes $\{m_f,m_c,m_h,m_i\}$, corresponding to UR_1^S=$\{m_f,m_h\}$.

When a new record (p_{35},m_c) is added to STreatments, UR_1^S becomes $\{m_f,m_h,m_c\}$. Thus, constructing the new NUR_1^S will start by following exactly the same steps resulting in NUR_1^S becoming $\{m_f,m_c,m_h.m_i\}$. Next, the third value in UR_1^S (m_c) should be inserted into NUR_1^S with another noise value. However, since m_c already exists in NUR_1^S (as a noise value from the previous steps), therefore we insert two random values to guarantee that the final NUR_1^S will contain three real values and three noise values. This is achieved using the following query.

```
Q3: select SMID from SMedications
    where SMID not in ('mf', ' mc', ' mh', ' mi')
    and TS < ts_mc
    order by rand(seed3) limit 2
```

Assuming that the query returns $\{m_g,m_d\}$, then the final NUR_1^S becomes $\{m_f,m_c,m_h,m_i,m_g,m_d\}$, which is a superset of the previous version of NUR_1^S. □

Our noise selection scheme works well if the results for the same query stay unchanged or expand over time. However, if values can disappear from the result over time (because of the dynamic nature of data), then the scheme may not always be able to generate consistent noise values every time the query is issued. Thus, if (1) the same translator *T* is used for the same query before and after its result has shrunken, and (2) *T* happened to be storing the old values it received for that query; then *T* might be able to eliminate some of the noise values, and hence the privacy of the query result can be at risk. Although, typically, this event will not occur frequently, it would be interesting to investigate the extension of our scheme to handle this situation. The particular challenge is that solving this problem may require the server to keep track of all queries and results it has served in the past, which is not always practical.

Fortunately, there is a wide class of applications where records are mostly appended rather than deleted or updated. Consider healthcare records, maintenance records, crime databases, retail transactions, and financial transactions to name just a few. Our protocol enables peers running any of such applications to exchange queries and results in a completely privacy-preserving manner. For other types of applications, where data is more likely to be updated or deleted, our protocol can still provide a fair degree of privacy, but with a small probability that certain query results may get disclosed over time. This can be useful for applications with data that is not highly sensitive. For example, we can think of a PDMS, where every peer is running a social network application, and through the PDMS it can allow its users to access their friends' lists on the other social networks.

V. Protocol Analysis

A. *Security Analysis*

In this subsection, we investigate the security of the protocol according to two models: (1) the *semi-honest* model, where all peers follow the protocol properly, but will try to gain additional information than they need if the protocol allows them to, and (2) the *malicious model*, where peers may deviate from the protocol to launch attacks against other peers. We will first present a theorem showing that in the semi-honest model, our protocol complies with the K-protection requirement. Then we will consider several possible attacks, assuming a malicious model, and discuss how our protocol is resilient to them.

Theorem 5.1. *Assuming a semi-honest model, the proposed privacy-preserving protocol complies with the K-protection requirement.*

Proof. During a single run of the protocol (call it $\wp$) to answer a query, each intermediate translator T_h, $h \in [1,n]$ only receives three messages: a Query_Msg, a Mapping_Req_Msg, and a Decryption_Req_Msg. These three messages represent the information leaked by the protocol to T_h., or $\mathfrak{R}(\wp, T_h)$. First, it is clear that T_h cannot infer any information about the query result from the Query_Msg. Second, since all the values in the Decryption_Req_Msg are encrypted using the client's key, then T_h cannot infer any of these values. Third, for each attribute a_j in the query result, the number of a_j values received in the Mapping_Req_Msg is equal to the minimum of $k_j \times |UR_j^S|$ and $|D_j^S|$ (Algorithm 1, line 1.19). Thus, since the number of unique values of a_j in the query result is $|UR_j^S|$, then T_h can be confident that a received a_j value belongs to the query answer with probability: $max(1/k_j, |UR_j^S|/|D_j^S|)$. This shows that the proposed protocol complies with the K-protection requirement (see Definition 3.1). □

We now discuss seven possible attacks against the protocol in the malicious model. The first four attacks are related to the privacy of the query result, while the following three attacks are related to the privacy of the mapping tables.

1) Query Replay Attack : During the query delivery phase of the protocol, a translator T will learn the client's query according to its own schema, Q^T. In a query replay attack, T will re-issue the same query, Q^T, several times, hoping that each time it may learn more information about the result. However, the discussion in Section IV has shown that for the same result, the selected noise values are always the same. Also, for a new result that is a superset of a previously encountered result, the set of real values in that new result along with the noise values will also form a superset of the real and noise values from the previously encountered result. Thus, the replay attack cannot be launched.

2) Known Result Attack: In this attack, a translator T may send queries to S whose results are already known to T, in an attempt to learn how S selects noise values. In particular, T can attempt to build a lookup table for every attribute a. Given a certain instance of UR_a^T and an instance value for k_a, the table gives the corresponding noise values that T would receive mixed with UR_a^T. By UR_a^T, we refer to the unique values of a in the query result based on T's schema, and by k_a, we refer to the value corresponding to a in the query's K vector. T can populate a cell in this table by (1) sending a query to S whose result is known to have a UR_a^T instance identical to the UR_a^T instance corresponding to the cell being populated, and (2) specifying a value for k_a equal to that cell's corresponding k_a value. The cell will then be populated by the noise values returned by S, which can be easily isolated since the real values are already known. Consider that the number of cells T manages to populate in this way is $N_{populated\_cells}$. If D_a^T is a's domain in T, then a UR_a^T instance can be any *ordered* subset of D_a^T. Order matters because the selected noise values depend on the timestamp ordering of the values in UR_a^T. Thus, knowing that k_a can take any value from 1 to $|D_a^T|$, the total number of cells in the lookup table is given by

$$N_{all\_cells} = |D_a^T| \sum_{i=1}^{|D_a^T|} \frac{(|D_a^T|)!}{(|D_a^T| - i)!} \qquad (1)$$

Clearly, this number can be very large as the size of D_a^T increases. Consequently, assuming a uniform distribution for cell population, the probability that any given cell is populated, which is given by ($N_{populated\_cells}/N_{all\_cells}$), will be very small. We now explain how T can use the lookup table to isolate the noise values when the real values are not previously known. During phase II of the protocol, T receives a Mapping_Req_Msg whose corresponding NUR_a^T (list of unique noise and real values of a) contains $k_a \times |UR_a^T|$ values. Thus, for every possible instance of UR_a^T within NUR_a^T, T can initially assume that the remaining $(k_a\text{-}1) \times |UR_a^T|$ values are noise values. Then, T can verify if this assumption holds by checking the cell in the lookup table, which corresponds to the UR_a^T instance at hand and the specified value for k_a. For T to succeed in identifying the noise values, this assumption must hold for only one instance of UR_a^T within NUR_a^T, and fails for all the other instances. The total number of possible instances of UR_a^T within NUR_a^T, or equivalently, the number of cells that must be checked, and hence must be populated is given by

$$N_{required\_cells} = \frac{(k_a \times |UR_a^T|)!}{(k_a \times |UR_a^T| - |UR_a^T|)!} \qquad (2)$$

It follows that the probability of success for this attack is given by

$$\Pr(attack\_success) = \left(\frac{N_{populated\_cells}}{N_{all\_cells}} \right)^{N_{required\_cells}} \qquad (3)$$

This probability is typically extremely small since N_{all_cells} is much larger than $N_{populated\_cells}$, and $N_{required\_cells}$ is also a very large number. However, only when $|UR_a^T|$ is very small, and k_a is also small, $N_{required\_cells}$ may not be large enough. For example, if $|UR_a^T|$=1, and k_a=2, then $N_{required\_cells}$=2. However, to address this problem, when a client C sends a query to S, in addition to specifying k_a within the K vector, it can also specify a minimum bound on $N_{required\_cells}$, or $N_{min\_required\_cells}$. This way, if after executing the query, S discovers that $N_{required\_cells} < N_{min\_required\_cells}$, then S should increase k_a appropriately, such that $N_{required\_cells} \geq N_{min\_required\_cells}$. This guarantees that the probability of success for this attack will always be negligibly small.

3) Privacy Relaxation Attack: Based on the protocol description given in Subsection III-B, a translator T_h, $h \in [1,n]$ may attempt to reduce the values in K before forwarding the query to $T_{(h+1)}$. This way, T_h can illegitimately relax the privacy requirement set by C. To guard against this attack, phase I of the protocol should be modified as follows. C should send the K vector directly to S and not to any translator, along with the names of the attributes requested in Q^C (which conform to C's schema). Moreover, every translator T_h, $h \in [1,n]$, should send to S the attribute name mappings it used for translating the query. At the end of phase I, S will be able to translate the attribute names it received from C to the attribute names of its own schema, using the mappings sent by the translators. Now, S can associate each attribute with its corresponding value in K without giving any opportunity for T_h to change such values.

4) Translators Collusion Attack: In this attack, several translators may collude together in an attempt to gain additional knowledge about the result. However, by analyzing the protocol, we find that the only inputs a translator gets related to the result are the Mapping_Req_Msg it receives from the server and the Decryption_Req_Msg it receives from the client. The Decryption_Req_Msg's are encrypted by the client's key, so no information can be inferred from them by the translators. Furthermore, the different Mapping_Req_Msg's contain different translations for the same set of real and noise values originally sent by the server. Therefore, if the translators collect all their received Mapping_Req_Msg's in a central location, this will still not allow them to identify any of the noise or real values, and hence their knowledge about the result will remain the same. In other words, this attack cannot be launched.

5) Mappings Correlation Attack: In this attack, the client may attempt to correlate the mappings $E_{(h+1)}(M_{(h+1)})$ and $E_h(M_h)$ sent by $T_{(h+1)}$ and T_h respectively (recall that these mappings map plaintext values to encrypted values) to find M_h, $h \in [2,n\text{-}1]$. The key idea is to try to match each plaintext T'_h value from the first mapping to its corresponding plaintext $T'_{(h-1)}$ value from the second mapping. This should result in obtaining M_h. However, the two mappings have different orderings, since they both have been randomly shuffled (Algorithm 2, line 2.6). Therefore, it is not possible to match corresponding values based on their positions in the mappings. Thus, launching this attack will not succeed.

6) Rich Client Attack: In this attack, a rich client C may decide to purchase all the mapping entries from a certain translator T. This way, it can clone T and provide the same translation services, which T previously offered. Thus, T can be deprived from potential future revenues. To see how such attack can be stopped, we first draw an analogy between the revenue model of T and the revenue model of carfax [22], a real company specialized in providing vehicle history reports to its customers. Similar to T, carfax would like to protect its database of vehicle history reports from getting largely disclosed to a single customer or even to a small group of customers. To address this type of attacks, carfax's customer agreement include the following statement: "Any commercial use of carfax vehicle history reports is strictly prohibited and any suspected commercial use will lead to the suspension of your account". The term "suspected commercial use" implies requesting reports at a rate that is higher than a certain threshold, or that matches more sophisticated patterns indicating misbehavior. The details of such patterns are beyond the scope of this paper. However, we emphasize that this type of attacks can be detected and stopped using methods that are already applied by established real-world businesses.

7) Black Market Attack: We refer by this attack to the situation where clients choose to form a black market to exchange mapping entries and thus negatively impact the revenues of the translators. Since it is not possible to completely stop this attack, we will experimentally show in Subsection VI-C how our protocol manages to limit the effect of this attack compared to other possible protocols.

B. *Cost Analysis*

If c_e is the cost of encryption/decryption, c_h is the cost of evaluating the hash function, and if we consider that the costs of performing these operations dominate other computation costs, then the overall computation cost for executing one query is given as follows

$$c_{comp} = \sum_{j=1}^{l} |UR_j^S| \times c_h + 3 \times n \times \sum_{j=1}^{l} |UR_j^S| \times c_e \qquad (4)$$

The first term in (4) corresponds to the hashing operations performed by S to obtain the seeds used to generate the noise values. From Section IV, S performs a hashing operation for every unique real value in R^S, whose count represents the summation in the first term. The second term corresponds to the series of encryption, decryption, and re-decryption operations performed by C, T_h, and C respectively for every unique real value in R^S, and for every $h \in [1,n]$. Note that the initial encryption step that resulted in the encrypted values sent in the Mapping_Resp_Msg of each translator T_h have been performed offline by T_h, rather than at query processing time. Thus, its cost is not accounted for in (4).

If b_q is the number of bytes in the query, b_p is the number of bytes in plaintext value, b_c is the number of bytes in a singly-encrypted value, and b_{cc} is the number of bytes in a doubly-encrypted value, then the overall communication cost for executing one query is given as follows

$$\begin{aligned} c_{comm} &= (n+1) \times b_q \\ &+ \sum_{j=1}^{l} |UR_j^S| \times b_p \\ &+ n \times \sum_{j=1}^{l} |UR_j^S| \times k_j \times b_p \\ &+ n \times \sum_{j=1}^{l} |UR_j^S| \times k_j \times (b_p + b_c) \\ &+ n \times \sum_{j=1}^{l} |UR_j^S| \times b_{cc} \\ &+ n \times \sum_{j=1}^{l} |UR_j^S| \times b_c \qquad (5) \end{aligned}$$

The terms in (5) represent the communication costs for the Query_Msg's, Result_Msg, Mapping_Req_Msg's, Mapping_Resp_Msg's, Decryption_Req_Msg's, and Decryption_Msg's respectively.

VI. Experiments

A. *Experimental Setup*

Our experimental study has two main goals.

- Studying the protocol's performance in a real system using real data.
- Studying the fairness properties of protocol.

For the performance study, the metrics we use are (1) the query total response time and (2) the total number of bytes transmitted over the network per query execution. We compare the performance of the privacy-preserving protocol (call it PPP) to that of the regular non-privacy-preserving protocol (call it NPPP). Moreover, we vary two parameters in this study: (1) the path length from the client to the server and (2) the K vector used to specify the K-protection privacy requirement.

To study the fairness properties of the protocol, we use the total revenue made by the translators as our metric. The revenue made when PPP is used is compared to that made when another protocol is used, which is a variant of PPP (call it PPP_V). PPP_V operates in the same way as PPP except that the Mapping_Resp_Msg's contain plaintext-to-plaintext value mappings rather than plaintext-to-encrypted value mappings; i.e. the client learns (and pays for) the mappings both for the real and noise values. The translators' revenue made with these protocols is also compared against the upper bound (UB) and lower bound (LB) of the fair range for the revenue (see Definition 3.2). The parameters varied in the fairness study experiment are: (1) the client behavior, which can be one of "no cache and no black market", "cache and no black market", or "cache and black market"; (2) the client cache size; and (3) the number of black market search attempts a client performs before resorting to purchasing a mapping entry from its owner translator.

We use a dataset obtained from a pet hospital chain to conduct our experiments. The original data set includes information about the demographics of pets, their hospital visits, diagnosed diseases, lab tests, prescribed medications, etc. The dataset contains information about over 45 million pets with hospitals spanning 40 US states. For the sake of the experiments, we modified the data set as follows. First, we partitioned the data based on the state information to obtain 40 different dataset partitions. Second, for each partition, we mapped the data into a smaller schema, similar to the ones shown in Figure 3, thus keeping only the pet medication information. We assumed that each partition represents the database of a peer in the network. In particular, we used six such partitions representing six different states in our experiments. Table 2 shows some statistics about these dataset partitions.

TABLE 2. STATISTICS ON THE DATA SETS USED IN THE EXPERIMENTS

State	# pets	# medications	Size of Treatments table	Av. # medications / pet
AL	25015	1934	350639	14.02
AR	10253	1392	156659	15.28
AZ	217688	2950	3724275	17.11
CA	427433	3435	8136616	19.04
CO	195067	3194	4139109	21.22
CT	18585	2193	375267	20.20

We implemented the Pohlig–Hellman commutative encryption algorithm [16] using the GMP C library for cryptography applications [23]. Although Hyperion is implemented in Java, we integrated it with the C library as our experiments showed that it is 10x faster than the corresponding Java implementation of the encryption algorithm. We ran each Hyperion peer on a Sun Blade 1000 with 2GB RAM and two 1.2GHz SPARC V9 processors. The peers were connected through a 100Mbps LAN. The DBMS we used is MySQL. Each Hyperion peer used one of the dataset partitions as its local database.

B. Studying the Protocol Performance

The setup of this experiment was based on the observation that the performance of each individual query in a PDMS depends on the path between the client and the server, rather than the network as a whole. To this end, we used six machines for this experiment. Thus, we were able to vary the path length from 3 to 6.

Every data point reported in the experiment was measured by running a workload of 100 random queries and taking the average of their measurements. A random query requests all medications for some randomly selected pet in the server peer.

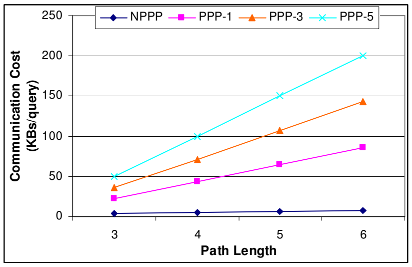


Figure 6. Effect of path length on the query response time

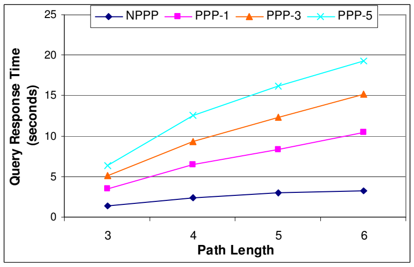


Figure 7. Effect of path length on the communication cost

Figures 6 and 7 show the query response time and the communication cost respectively for NPPP and PPP as the path length increases. In the figures, PPP- k_1 corresponds to PPP when K is set to (k_1), $k_1 \in \{1,3,5\}$. When K is set to (1), no noise values are inserted in the result, which implies that the result is not protected. However, the value of this special case is that the measured overhead only corresponds to the encryption/decryption operations performed by PPP. It is observed that response time is increased by a factor ranging from 2.5 to 3.3 for PPP-1 compared to NPPP, while the communication cost is increased by a factor ranging from 6.8 to 11.5 also for PPP-1 compared to NPPP. The increase in the communication cost is due to the larger number of bytes needed to represent the encrypted values. As indicated in Subsection V-B, increasing the values in K does not increase the encryption/decryption cost. Therefore, the higher overhead of PPP-3 and PPP-5 is only attributed to the inserted noise values. Comparing PPP-5 to NPPP, we find that the response time is increased by a factor ranging from 4.5 to 6, while the communication cost is increased by a factor ranging from 15.5 to 26.6. These factors are expected because the client explicitly requests to get k_1-times the original number of values in the result for privacy preservation. Generally, cryptography-based techniques incur an overhead in return for the benefits they bring. A good example is the work by Agrawal et al. [2], which also employs commutative encryption, but for a different purpose: finding shared information between private databases. In our context, we have verified that the overhead does not exceed a few extra seconds in query response time, which does not seem a high price for preserving privacy.

C. *Studying the Protocol Fairness*

In this experiment, we study the impact of having different client behaviors on the fairness of the protocols. We use a simple economic model, where the cost of a mapping entry is fixed for all peers. In reality, more sophisticated models may exist (e.g. based on supply and demand). However, this simple model still serves the purpose of showing the different levels of fairness achieved by the protocols.

It is obvious that if clients use no caches, then with PPP, they will have to purchase every mapping entry needed from its owner translator. Thus, the translators' revenue will be identical to UB. With PPP_V, the translators' revenue will be much higher depending on the setting of *K*, and thus PPP_V is unfair to the clients in this case. In what follows, we focus on the cases when the clients choose to use caches without forming a black market, or use caches and form a black market. When using caching, clients can have different cache sizes. Also, when the black market is used, clients may choose different number of attempts to search in the black market before resorting to purchasing a mapping entry for its regular price from the owner translator.

To study the effect of all such parameters, we developed a trace-based simulator that simulates the protocols PPP and PPP_V, where *K* is set to (4) for both of them. The simulator allows us to consider a large network while being able to easily vary the above parameters. In our experiment, we simulated a network of 50 peers. The network topology was based on a random graph, were the probability of having a link between any pair of peers was set to 0.05. For simplicity, we assumed that the local database for each peer is similar to one of the dataset partitions in Table 2. In particular, all local databases have distributions similar to that of the AR partition. But of course, it is still assumed that the values they store are different, and thus mapping tables are still needed.

The client caches follow an LRU replacement scheme. The black market is operated as follows. There is a central bulletin board where all clients advertise the mapping entries they have in their caches. An advertisement states that a given client peer has a given number of mapping entries from a given translator. Whenever a client is in need for a mapping entry it gets a list of the other client peers who advertised that they have the highest number of mapping entries from the desired translator. The size of the list is decided by the requesting client, which then contacts the peers on the list in descending order of the number of mapping entries they have. This number of peers contacted represents the number of black market search attempts.

Each data point reported in this experiment represents a simulation run, in which a trace of 10,000 random queries are executed in the network. A random query requests all the medications for a randomly selected pet in the server's database, given that both the client and server peers are also randomly selected. The network links are assumed to have equal weights.

Figure 8 shows the effect of varying the size of client cache on the revenue collected by all the translators in the network when clients use caching and no black market. It is clear form the figure that with PPP, the revenue is within the fair range. In contrast, with PPP_V, it is much higher than the upper bound of the fair range, which implies that the PPP_V is unfair to the clients.

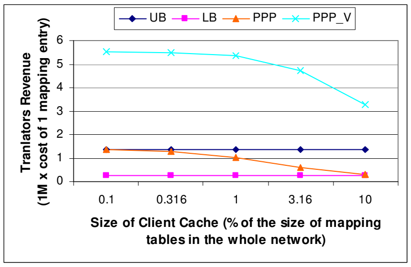


Figure 8. Effect of cache size on protocols fairness when clients use caching and no black market

Figure 9 also shows the effect of the cache size on the translators revenue, but this time, when the clients use caching in addition to the black market. Clients make four search attempts (equivalent to 8% of the total number of peers) in the black market before purchasing a mapping entry from its owner. Again, PPP is shown to be generally fairer than PPP_V. The effect of the black market is also clear in this figure, as the translators revenue is significantly lower in this case compared to the case when the black market is not used. Expectedly, the black market effect increases as the cache sizes increase, since more mapping entries will be available in the black market.

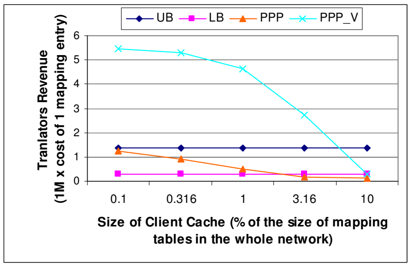


Figure 9. Effect of cache size on protocols fairness when clients use caching and black market

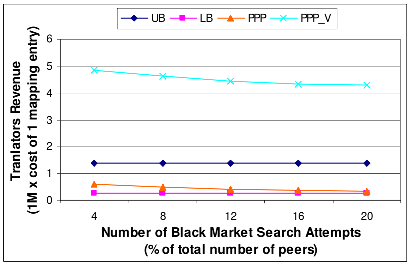


Figure 10. Effect of number of black market search attempts on protocols fairness

Figure 10 shows the impact of changing the number of black market search attempts on the translators' revenue. The clients' cache size is set to be 1% of the total size of mapping tables in the whole network. The results here too show that PPP is substantially fairer than PPP_V.

VII. Related Work

The application of P2P approaches to database systems has attracted considerable attention in the recent years. Gribble et al. [8] showed some benefits that the database technology can bring to the P2P research. They also addressed the data placement issues in P2P systems from a database perspective. PeerDB [15] is a P2P distributed data sharing system, which supports fine grained content-based searching and does not rely on a shared schema. It employs information retrieval approaches, such as keyword search, to find peers having data relevant to the user's query. It also uses mobile agents to be able to perform operations at peer sites. coDB [7] is another P2P database system with techniques for searching and updating peer databases. In coDB, mappings are defined in terms of GLAV *coordination rules*. Halevy et al. introduced the Piazza PDMS [9, 19, 20] to address the problems of defining a language for schema mappings, reformulating queries based on these mappings and automating the schema matching process in order to assist the user in constructing the mappings. In [11], Kementsietsidis et al. addressed the issue of mapping data, rather than schemas, in P2P data sharing systems. They propose treating the tables used in mapping the data from different peers as *mapping constraints* in order to be able to reason about them. In particular, they propose algorithms for inferring new mapping tables and determining the consistency of a set of mapping constraints. This whole body of work did not consider the security issues that need to be addressed in PDMSs.

Miklau et al. [14] proposed a framework for controlled sharing of XML data. The framework includes high-level access control policies, a logical model for protecting a document tree, and encryption techniques to construct an XML document that enforces the policies. This work however assumed a shared schema among all peers. Agrawal et al. [2] attempted to formalize the notion of limited information sharing across private databases. They focused on providing secure protocols for the computation of the intersection and equi-join of two sets in two different databases. Their work did not consider any semantic mappings between the sources, and therefore no translators were involved between the two communicating databases. Interestingly, their protocol also relies on commutative encryption.

A great deal of research is occurring in the area of k-anonymity [e.g. 1, 4, 12, 13, 18, 21]. A key difference between our work and this body of work is that we do not apply any generalization or suppression to the data in the way k-anonymity techniques do; instead we perturb the data by introducing noise values which are not part of the query answer. Another body of work addresses the problem of output perturbation in statistical databases [e.g. 5, 6]. However, similar to the work on k-anonymity, this body of work targets privacy preservation while maximizing data utility, for the purposes of statistical analysis and data mining. In contrast, our goal is to limit the utility of the query result for the translators, such that they can only translate the values, while learning the minimum information possible about the result. To the best of our knowledge, there has not been prior work addressing the privacy and fairness issues in PDMSs similar to this work.

VIII. Conclusions

In this paper, we introduced a novel privacy-preserving protocol that protects both query results and mappings in a PDMS, while maintaining fairness among peers. This protocol is based on noise insertion and commutative encryption methods. Our implementation on top of the Hyperion PDMS, our security and cost analyses of the protocol, and our extensive experimental study using real data from the healthcare domain showed promising results for our approach.

References

[1] C. Aggarwal. On k-Anonymity and the Curse of Dimensionality. In *VLDB*, 2005.

[2] R. Agrawal, A. Evfimievski, and R. Srikant. Information sharing across private databases. In *SIGMOD*, 2003.

[3] P. Andritsos et al. Kanata: adaptation and evolution in data sharing systems. In *SIGMOD Recorod* 33(4):32-37, December 2004.

[4] R. J. Bayardo. Jr. and R. Agrawal. Data privacy through optimal k-anonymization. In *ICDE*, 2005.

[5] A. Blum, C. Dwork, F. McSherry, and K. Nissim. Practical privacy: The suLQ framework. In *PODS*, 2005

[6] I. Dinur and K. Nissim. Revealing information while preserving privacy. In *PODS*, 2003.

[7] E. Franconi, G. Kuper, A. Lopatenko, I. Zaihrayeu. Queries and Updates in the coDB Peer to Peer Database System. In *VLDB'04 Demonstration session*, 2004.

[8] S. Gribble, A. Halevy, Z. Ives, M. Rodrig, and D. Suciu. What can databases do for peer-to-peer. In *WebDB*, 2001.

[9] A. Y. Halevy, Z. G. Ives, D. Suciu, and I.Tatarinov. Schema mediation in peer data management systems. In *ICDE*, 2003.

[10] A. Kementsietsidis and M. Arenas. Data Sharing Through Query Translation in Autonomous Sources. In *VLDB* 2004

[11] A. Kementsietsidis, M. Arenas, R. J. Miller. Mapping Data in Peer-to-Peer Systems: Semantics and Algorithmic Issues. In *SIGMOD,* 2003.

[12] K. LeFevre, D. J. DeWitt and R. Ramakrishnan. Incognito: Efficient Full-Domain K-Anonymity. In *SIGMOD*, 2005

[13] A. Meyerson and R. Williams. On the complexity of optimal k-anonymity. In *PODS*, 2004.

[14] G. Miklau and D. Suciu. Controlling Access to Published Data Using Cryptography. In *VLDB*, 2003.

[15] W.S. Ng, B. C. Ooi, K.L. Tan, A. Zhou. PeerDB: A P2P-based System for Distributed Data Sharing. In *ICDE*, 2003

[16] S. Pohlig and M. Hellman. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. IEEE Transactions on Information Theory, 24:106–110, Jan 1978.

[17] P. Rodriguez-Gianolli et al. Data Sharing in the Hyperion Peer Database System. In *VLDB*, 2005

[18] P. Samarati and L. Sweeney, Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. In *Proc. of the IEEE Symposium on Research in Security and Privacy*, 1998.

[19] I. Tatarinov and A. Y. Halevy. Efficient query reformulation in peer-data management systems. In *SIGMOD*, 2004.

[20] I. Tatarinov et al. The Piazza Peer Data Management Project. In *WWW*, 2003.

[21] C. Yao, X. S. Wang, S. Jajodia. Checking for k-Anonymity Violation by Views. In *VLDB*, 2005.

[22] http://www.carfax.com

[23] http://www.gmplib.org