HKU CS Tech Report TR-2011-12

Lightweight Privacy-Preserving Peer-to-Peer Data Integration

Ye Zhang * Wai-Kit Wong† S.M. Yiu† Nikos Mamoulis†
David W. Cheung †

August 28, 2012

Abstract

Peer Data Management Systems (PDMS) are an attractive solution for managing distributed heterogeneous information. When a peer (client) requests data from another peer (server) with a different schema, translations of the query and its answer are done by a sequence of intermediate peers (translators). There are two privacy issues in this P2P data integration process: (i) answer privacy: no unauthorized parties (including the translators) should learn the query result; (ii) mapping privacy: the schema and the value mappings used by the translators to perform the translation should not be revealed to other peers. PPP [7], is the first protocol proposed to support privacy-preserving querying in PDMS. However, PPP suffers from several shortcomings. First, PPP does not satisfy the requirement of answer privacy, because it is based on commutative encryption; we show that this issue can be fixed by adopting another cryptographic technique called oblivious transfer. Second, PPP adopts a weaker notion for mapping privacy, which allows the client peer to observe certain mappings done by translators. In this paper, we develop a lightweight protocol, which satisfies mapping privacy and extend it to a more complex one that facilitates parallel translation by peers. Furthermore, we consider a stronger adversary model where there may be collusions among peers and propose an efficient protocol that guards against collusions. We conduct an experimental study on the performance of the proposed protocols using both real and synthetic data. The results show that the proposed protocols not only achieve a better privacy guarantee than PPP, but they are also more efficient.

1 Introduction

Peer Data Management Systems (PDMS) have become popular in the recent years [19, 23, 27], because they enable the management of heterogeneous data in a decentralized fashion. Real-life systems include Hyperion [23], PeerDB [19] and BestPeer[1]. In a PDMS, each peer owns or hosts a heterogeneous database. Due to the dynamic nature of the system, it is infeasible for the peers to agree on a global schema. Translations, in the form of mappings, are required for the communication between two peers. An example is shown in Figure 1; two peers hold two medical databases that have different schemas. Differences may exist in attribute names (e.g., ‘Name’ in Peer 1, ‘Patient’ in Peer 2) or in object values (e.g., ‘LC’ in Peer 1, ‘Lung Cancer’ in Peer 2). If Peer 1 knows the mappings of attributes and values of its database to the database of Peer 2, then Peer 1 can send queries to Peer 2 and translate the answers.

Two peers are said to be *acquainted* if there are direct mappings between them, i.e., one peer can translate the schema of the other.[2] By adding an edge between two acquainted peers (nodes), we can form an acquaintance graph [26, 4], which captures the feasible data flow in a P2P network. If a peer (*client*) wants to issue a query to another peer (*server*), it first finds a path to the server in the acquaintance graph.

*Pennsylvania State University, USA. Email: yxz169@cse.psu.edu. Part of this work was conducted while the author was with the University of Hong Kong.

†University of Hong Kong, Hong Kong. Email: {wkwong2,smyiu,nikos,dcheung}@cs.hku.hk

[1]http://www.bestpeer.com

[2]Two peers with identical schemas are trivially acquainted.

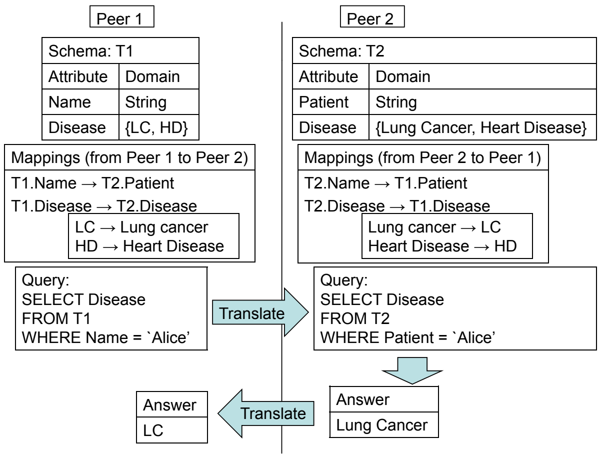

Figure 1: An example of query and answer translation between two peers with different schemas.

The intermediate peers on this path (*translators*) provide the translation service for the query and its answer. Thus, the query processing framework in a PDMS allows communication between peers even if they do not have direct mappings.

In this paper, we consider applications, where the privacy of the peers during this process must be protected. Consider the case where some peers are hospitals, which share information about their patients in the PDMS so that an authorized party (e.g., a registered doctor) can issue queries on the shared data. For example, the doctor asks from the client peer for the medications received by a patient at a server peer. During the process, a translator can observe the query answers; thus, the privacy of the patient is breached. The need for privacy in PDMS when used by healthcare applications is highlighted by a recent NIH report [18]. Besides query answers, which must be protected from the translators, the schema and value mappings owned by the translators should also be protected from other peers. Generating mappings between two peers requires specialized knowledge on the schemas of the two peers; this is an expensive process that involves significant human effort. Thus, it is fair to allow translators to charge clients for the translation service. This encourages more pairs of peers to establish accurate mappings between them and eventually improves the connectivity of the P2P network. At the same time, we need to protect the mappings of peers so that they are hidden from others while the translations of queries and answers can still be performed[3]. In other words, the query issuer and the server should only observe the query and answer in their own context and not obtain any mappings between peers.

These privacy issues in a PDMS have been recently considered in the pioneering work of [7]. A new privacy notion called *k-protection* is proposed.[4] The idea is similar to k-anonymity [25]: a translator should not determine whether a value belongs to the query result with certainty greater than $\frac{1}{k}$. Besides, the client is allowed to view only the mappings related to the query answer to protect mapping privacy. This requirement is referred to as *fairness*. A novel query processing protocol, called PPP, is proposed in [7] to address k-protection.

PPP makes use of two techniques, *fake answer injection* and *commutative encryption*, to solve the problem. The server injects noise to the query answer to confuse the translators while the client uses commutative encryption to retrieve the necessary mappings for the query answer from the translator. Figure 2 shows an example of PPP. For the ease of discussion, we assume there is only one translator between the client and the server. The server first sends the answer (in the server's schema) to the client (Step 1 of Figure 2) and then issues a *mapping request* that contains the answer (assume there is one value only in answer) and $k-1$ additional fake values to the translator (Step 2). Based on this mapping request, the translator

[3]Note that composition of mappings should also be protected, otherwise a client can avoid the charges of some intermediate translators.

[4]k-protection is not the strongest possible privacy requirement [12] but it offers a reasonable privacy protection while allowing efficient computation of queries;it is used in many applications like location-based services [31]. To strike a balance between privacy and efficiency, we adopt k-protection in our study.

retrieves the mappings, encrypts them, then sends the list to the client (Step 3). Note that the client cannot see the mapped values, but he is able to pick the corresponding encrypted value from the list. In order to securely decrypt the value with the help of the translator, commutative encryption is used. The client selects only the encrypted answer of the query and encrypts it using his own key and sends this double encrypted value to the translator (Step 4). The translator has no way to find out what value the client has selected. Commutative encryption enables the client to obtain the original value by applying decryption in any order independent to the encryption order. Thus, the translator can still apply the decryption procedure on the double encrypted value although he cannot understand the decrypted value. Finally, the client applies his own decryption again to obtain the original value (Step 5).

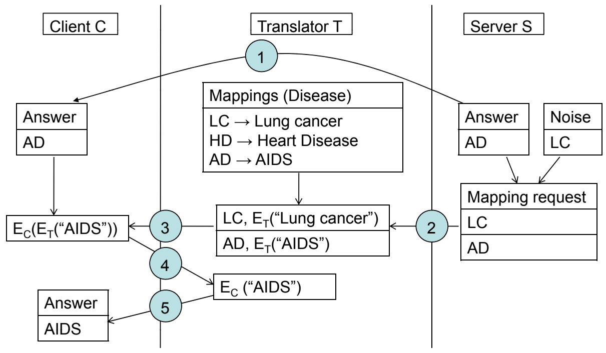

Figure 2: Execution of PPP with $k = 2$. The messages between peers are: (1) S $\to$ C: query result in the context of S. (2) S $\to$ T: mapping request. (3) T $\to$ C: encrypted mappings. (4) C $\to$ T: double-encrypted mapping. (5) encrypted mapping that can be decrypted by C.

1.1 Contributions and Outline

The security of PPP relies on the security of the commutative encryption. The most popular commutative encryption scheme (also used in the implementation of [7]) is Pohlig-Hellman [21]. As shown in [15] and independently in Section 3 and our technical report [32], Pohlig-Hellman can easily be attacked, thus PPP is not secure in practice. To our knowledge, there is no secure commutative encryption scheme in practice. Our first contribution (POT and POT-opt protocols, Section 3) is to fix this security issue of PPP by replacing commutative encryption by *oblivious transfer* (OT), a well-developed construct with strong security guarantee, that allows the client to obtain the necessary mappings without letting the translators know the selected items by the client. We remark that the OT protocol that we describe in this paper is customized to PDMS and is more efficient than the OT protocol in general case. However, even with this fix, we note that the PPP framework itself has several drawbacks:

1. *Poor protection of mapping privacy.* PPP does not completely protect mapping privacy: during query evaluation, several mappings of the translators are revealed to the client. A protection scheme that does not reveal any mappings is desired.

2. *Low efficiency.* PPP requires a large number of heavy cryptographic operations and thus the entire protocol has a high execution cost. For example, translating a query result with 20 values requires almost 20 seconds, as reported in [7], when the client and server are located 9 peers apart and $k = 5$.

3. *High communication cost.* The client communicates with each translator to obtain the required mappings using 3 rounds of communication. In addition, the use of encryption typically increases the message sizes. For example, Pohlig-Hellman with n-bit key generates ciphertexts of size $O(n)$ regardless of the original message size.[5]

[5]The original message, however, should be less than 1024 bits; otherwise, we have to break the message into two or more blocks.

Our second contribution is to address the above issues, by developing a *simpler* and *more secure* protocol PD, which does not rely on cryptographic operations, making it significantly more efficient than PPP (Section 4). We also develop a parallel version of our protocol (IMP protocol, Section 5), which is suitable for the case where large amounts of data need to be translated. Our third contribution is to extend the study to consider collusion, a stronger adversary assumption, and develop a novel lightweight protocol PC that achieves privacy under this assumption (Section 6). Finally, we conduct extensive experiments on both synthetic and real data to evaluate our solution (Section 7). The results show that our methods are more efficient than the state-of-the-art protocol [7] and at the same time they provide a stronger privacy guarantee.

2 Background and problem definition

In this section, we provide a formal definition for the problem of preserving privacy in a PDMS, give some background on security techniques, and describe the PPP [7] protocol.

2.1 Querying in PDMS

When a query is issued in a PDMS, peers may take one of following three roles: 'client' (C), 'server' (S) and 'translator' (T_i where i is a positive integer). C issues a query to be answered by S and T_i is an intermediate peer who performs translations. Without loss of generality, we assume that there are $t \geq 0$ translators and the path from C to S is $\{C, T_1, T_2, ..., T_t, S\}$. Each two consecutive peers T_i and T_{i+1} on the path from C to S own the mappings for translating to each other's schema; i.e., T_i can translate the query/answer from his own schema to T_{i+1}'s schema and vice versa. There are two different types of mappings: (i) attribute-to-attribute mappings (e.g., `fname` and `firstname` are attribute names used by two different peers to store first names of patients); (ii) value-to-value mappings (e.g., 'heart disease' is encoded as d01 at peer A and as d05 at peer B). Note that attribute-to-attribute mappings do not involve translation of sensitive data in the query result; therefore, in this paper, we consider value-to-value mappings only.

2.2 Privacy-preserving querying in PDMS

We assume that the query answer contains sensitive information about individuals; therefore it should only be accessed by authorized parties (in our case, the client C). All translators T_i are assumed to be *semi-honest*, i.e., they provide the correct messages for the protocol and follow the protocol properly but they are curious to obtain more information based on the messages they obtained in the protocol. Let Q^X be the query and R^X be the query answer in the context of the peer X. The goals of the translations are (i) the query Q^C issued by C is translated to Q^S; (ii) the query answer R^S of the query Q^S is translated to R^C. Like [7], we assume that a query does not contain sensitive information; thus, query translation (the first goal) can be achieved as in a non-privacy preserving protocol (i.e., the query trivially travels in the path from C to S and gets translated on the way)[6]. On the other hand, we require that the privacy of the answer is protected (**answer privacy**); i.e., no translator can learn the query result. A formal privacy requirement for this purpose, called k-protection, was proposed in [7]:

Definition 1 (k-protection [7]) *Let R^X, D^X, UR^X be the query answer, domain of the query answer, and the set of unique values in the query answer respectively in the context of X. Let $\Re(\pi, X)$ be the information observed by the peer X during the execution of the protocol π. A protocol π is said to provide k-protection if for each T_i, $i \in [1, t]$, $\Pr[v \in R^{T_i} \mid \Re(\pi, T_i)] \leq \frac{1}{k}$ for all $v \in D^{T_i}$.*[7]

Apart from the client's privacy concerns on the query answer, for fairness, the mappings of translators should also be protected from the client (**mapping privacy**) [7]. The mappings are important assets of translators: they allow them to charge clients for translation and they should not be revealed to clients.

[6]If the query needs to be protected, it can be translated in a similar fashion as the query answer.

[7]In [7], k is at most $\frac{|D^S|}{|UR^S|}$ as it is assumed that $|UR^S|$ and $|D^S|$ are not sensitive and can be leaked to the translators. Therefore, $\frac{|UR^S|}{|D^S|}$ acts as a default upper bound.

2.3 Security Techniques and Basic Concepts

A symmetric key encryption consists of three (probabilistic) algorithms (Gen, E, D). Given a security parameter (e.g., 80), Gen algorithm will generate a key $\mathcal{K}_T$ for every user T. Using $\mathcal{K}_T$, T can encrypt a message m to ciphertext $c = E(\mathcal{K}_T, m)$ (simply denoted by $c = E_T(m)$) and decrypt a valid ciphertext c to $m = D(\mathcal{K}_T, c)$ (or simply $m = D_T(c)$). Loosely speaking, a good encryption scheme will prevent adversaries from learning any information on m (other than the length $|m|$) from its ciphertext.

A symmetric key encryption scheme is *commutative* if for any two users S, T and message m, we have $E_T(E_S(m)) = E_S(E_T(m)) = c$ and $D_S(D_T(c)) = D_T(D_S(c)) = m$. Loosely speaking, commutative encryption is oblivious to the order of encryption/decryption.

We now provide some background on modular arithmetic concepts, used in this paper. For two integers $a, b \in \mathbb{N}$, we say that $a \mod n = b$ if and only if $b \in [0, n-1]$ and $a - b = kn$ for some integer k. Two integers a, b are *co-prime* if and only if their greatest common divisor $gcd(a, b)$ equals 1. A known fact is that there exists a $c \in [1, n-1]$ such that $ac \mod n = 1$ if and only if $gcd(a, n) = 1$; c is called the *multiplicative inverse* of a (mod n) and c is unique. The *Extended Euclidean Algorithm* is an efficient algorithm for computing c from two integers a, n such that $gcd(a, n) = 1$.

2.4 The PPP protocol

Figure 2 illustrates PPP. If there are x values in the answer set, the server will insert $(k-1)x$ random values as noise to provide k-protection. Then the answer set is randomly shuffled to prevent immediate identification of the noise. In addition, the noise for the same query issued at different times should be the same; otherwise, it may be possible for an attacker to identify the common values of multiple answer sets of the same query, as the query answer. To simplify the discussion, let $x = 2$ and $k = 2$. Let the true answer set be (m_1, m_2) and the answer set with noise be $(m_\alpha, m_2, m_1, m_\beta)$. The *true* answer set is sent to the client directly. The answer set with noise (mapping request) is sent to a translator T for translation. Let $(m'_\alpha, m'_2, m'_1, m'_\beta)$ be the translated values by T. To prevent the client C from learning unnecessary mappings, T encrypts them using its private encryption key. Let $E_T(x)$, $E_C(x)$ be the encryption functions on value x used by T and C respectively. The encrypted result $((m_\alpha, E_T(m'_\alpha)),$ $(m_2, E_T(m'_2)), (m_1, E_T(m'_1)),$ $(m_\beta, E_T(m'_\beta)))$ is sent to C. C selects the two encrypted mappings $((m_1, E_T(m'_1)),$ $(m_2, E_T(m'_2)))$ (i.e., the true answer set) and generates the ciphertexts $(E_C(E_T(m'_1)),$ $E_C(E_T(m'_2)))$ which are sent to the translator T. T decrypts the ciphertexts to $(E_C(m'_1), E_C(m'_2))$, based on the fact that $E_C(E_T(M)) = E_T(E_C(M))$ for *commutative* encryption functions E_C, E_T. Finally C decrypts m'_1 and m'_2 and obtains the answer set.

3 Fixing the PPP protocol

PPP is proved to be secure in [7], assuming that commutative encryption is secure. However, existing commutative encryption schemes, like Pohlig-Hellman [21] and SRA[24], do not provide formal proofs of security [29] and may lead to security breaches in practice, as shown in [15]. Specifically, Pohlig-Hellman leaks the information whether x is quadratic residue (mod n) or not (a property similar to whether a number is odd or even). For example, if the query answer is x which is quadratic residue but the $k-1$ noise results added are not, then the adversary can directly identify which one is in the answer set. In [32], we provide more details about this attack. Therefore, we can claim that when instantiating PPP with the existing commutative encryption schemes, k-protection cannot be enforced.[8] In the following, we provide a fix to PPP, which replaces commutative encryption by an oblivious transfer protocol.

3.1 Oblivious transfer

Oblivious transfer (OT) [22] is a well-developed probabilistic approach, allowing a party to retrieve information from another party, in which the sender does not know what is retrieved by the receiver. In a

[8]Of course, in the above example, on could enforce all added noise to be/form quadratic residue to avoid such an attack. However, we still cannot prove that Pohlig-Hellman (and SRA etc.) scheme will not leak any other forms of information **nikos: what other information?**.

nutshell, the server owns a number of data items, each with a unique index and the receiver wants to retrieve the data with index i from the sender. An OT protocol allows the receiver to obtain the corresponding data without seeing other data; at the same time the sender cannot learn i. There are different implementations of OT with varying levels of security guarantee and efficiency. In this paper, we employ the framework proposed in [5] and instantiate it with Chaum's signature scheme [3]. The combined algorithm is secure under the semi-honest model and is relatively simple and computationally efficient (optimized with two rounds of communication). A high-level description is shown next; the mathematical details can be found in Appendix B.

The goal of OT in our protocol (POT protocol) is to allow the client C to retrieve the necessary mappings from a translator T. For each mapping entry $x \to y$, T preserves the left part x in plain form. This allows C to choose the required mappings. The right part y is encrypted by a special encryption function. The goal of the encryption is to prevent C from observing unnecessary mappings while C can decrypt some entries that he has chosen. In the encryption of y, we include x in the parameters so that the encrypted value of y depends on both x and y. We denote the encrypted value of y by $E_T(x, y)$. A customized key in decrypting $E_T(x, y)$ is needed and is composed of (i) the secret key of the encryption algorithm E_T and (ii) the choice of x. We denote the customized key for decrypting $E_T(x, y)$, $\mathcal{K}(x)$. T sends to C the list of encrypted mappings, in the form of $x \to E_T(x, y)$. To decrypt $E_T(x, y)$, C has to ask T for decryption keys for selected entries. However, directly informing T_i the selection of mapping entries violates the privacy of the query answer. Suppose C desires the mapping of x ($x \to E_T(x, y)$). C encrypts his selection by $E_C(x)$ and sends to T $E_C(x)$. In our implementation, E_T and E_C are specially designed algorithms so that T can transform $E_C(x)$, by using his private key of E_T, to the encrypted decryption key for y $E_C(\mathcal{K}(x))$. In the transformation process, T cannot observe the selection of C since he works on the encrypted domain only. The encrypted key is sent back to C. C can get the decryption key $\mathcal{K}(x)$ and thus recovers y. The decryption key cannot be used on other encrypted mapping entries since the key is specific to entry $x \to y$ only.

Theorem 1 *The POT protocol satisfies k-protection.*

Proof: The proof is shown in Appendix B.1.

3.2 Optimizations

In our implementation, we apply some optimizations to reduce the computational cost of OT, which are shown in our empirical study to significantly improve efficiency.

3.2.1 Pre-computation of encrypted mappings

Each encrypted mapping entry deterministically depends on the value of the mapping and the mapping owner's key. In our implementation, the translators pre-compute and cache the encrypted mappings, in order and avoid their online computation whenever they are requested.

3.2.2 Algorithmic Speeding up Techniques

A major component in our oblivious transfer protocol is the exponential cipher. This involves computing modular exponentials of large values (a typical message length is 1024 bits). These operations are very expensive. One way to reduce the cost is to use Garner's algorithm [17]. The basic idea is the Chinese Remainder Theorem: computing $b = x^e \mod pq$ is equivalent to computing $b = [(x^e \mod q - x^e \mod p) \times (p^{-1} \mod q) \mod q] \times p + (x^e \mod p)$. The theorem reduces the computation of a modular exponentiation to two 512-bit modular exponentiations with some cheaper additions and multiplications.

Note that the cost of computing modular exponential r^e highly depends on the value of e, where e is part of the public key in our case. To reduce this cost, we set $e = 3$ at each translator. Setting $e = 3$ does not improve an adversary's knowledge, since the key is public[9]. Without knowing the factorization of n, an adversary cannot determine the private key by simply knowing that $e = 3$.

[9]In practice, RSA usually sets the public exponent e to be a small constant to support efficient computations.

4 An efficient alternative to PPP

Oblivious transfer helped us to fix a security hole of PPP. Still, the PPP framework suffers from the following problems: (i) incomplete protection of mapping privacy; (ii) high response time; (iii) high communication cost. To tackle the above issues, we develop a simple lightweight protocol, which does not rely on cryptographic operations and not only preserves k-protection but also protects mapping privacy. In this section we describe the new protocol and compare its security and efficiency with PPP.

4.1 Order-preserving translation

Our proposed protocol (denoted by PD, Privacy preserving and Direct simple method) **Nikos: explain what PD means!** operates as follows. After the server S computes the query answer, S carefully adds noise to it to form a mapping request. For a query answer with x distinct values to be translated, a noise of $(k-1)x$ values in the domain is added. Thus, the mapping request composes of kx values to be translated. In addition, the positions of values to be translated in the mapping request are randomly shuffled at S. The query answer is represented using the corresponding positions of values in the mapping request and the query answer is sent to C. For example, suppose that the query answer is (m_1, m_2, m_1) and the mapping request after random permutation is $(m_\alpha, m_2, m_1, m_\beta)$. The server S first sends directly to C an *answer key* $(3, 2, 3)$; i.e., the positions of the true answer values in $(m_\alpha, m_2, m_1, m_\beta)$. The noise generation process is the same as [7] to prevent query replay attack. That is, for two identical queries to S, exactly the same noise and permutation will be generated.

If the path from C to S is $\{C, T_1, T_2, \ldots, T_t, S\}$, the mapping request is transferred from S to T_t for translation. Each translator T_i translates all the values in the mapping request and forwards the translated mapping request to the next translator T_{i-1} until it reaches the client. Differently from PPP, each translator *does not permute* the positions of values in the mapping request. Eventually, the client C will receive the mapping request, which is a set of translated values under C's context. In our example, assuming that the path is $\{C, T_1, S\}$ and that T_1 translates each m_i to m'_i, C will obtain $(m'_\alpha, m'_2, m'_1, m'_\beta)$. With the query answer key $(3, 2, 3)$ that C has received from S, C recovers the query result (m'_1, m'_2, m'_1).

4.2 Security proof and cost analysis

In this section, we prove the security of the proposed protocol and analyze its cost.

Theorem 2 (Answer privacy) *The PD protocol satisfies k-protection.*

Proof: Each translator T_i receives a mapping request from T_{i+1} with kx values, where x is the number of distinct values in the query answer to be translated and k is the privacy parameter. Without the answer key from the server S, each value has the same probability ($\Pr = \frac{x}{kx} = \frac{1}{k}$) to be in the query answer, i.e., k-protection is enforced. In addition, every possible mapping request corresponds to a specific answer set and a specific permutation (by S), thus for the same query T_i receives the same mapping request. Therefore, T_i cannot learn anything about the query answers even if the same query is repeatedly sent from C to S.

Theorem 3 (Mapping privacy) *The PD protocol enforces mapping privacy.*

Proof: Each translator T_i observes only one message: a mapping request with kx values from translator T_{i+1} and no messages from any other peer. The mapping request received by T_i contains only a set of values in T_{i+1}'s context, therefore T_i learns nothing about T_{i+1}'s mappings.

Compared to PPP, our simple protocol offers better privacy protection because mapping privacy is fully protected. The only exception is that PPP has a better worst case protection on privacy mappings in case of collusion. In PPP, translators also permute the mapping request randomly at the price of letting the client learn certain mappings. This means that even when several parties collude together and exchange their knowledge to beach mapping privacy, PPP can protect one's mappings from being seen (except the entries that are already revealed to the client). In Section 6, we study the issue of collusion in detail and provide a complete lightweight solution that protects both answer privacy and mapping privacy in case of collusions.

Cost analysis. Our simple protocol is very efficient and has a low message communication cost. Let n be the number of distinct values to be translated, t be the number of translators, and k be the privacy parameter of k-protection. At the server, the preparation cost of the mapping request and the answer key takes $O(kn)$ time. The server sends two messages: (i) the mapping request to the neighbor translator T_t, which contains kn values; (ii) the answer key, which has the same size as the query answer. Each translator T_i takes $O(kn)$ times to translate the mapping request and forward the request to the next translator T_{i-1} (or the client). Compared to PPP, our protocol has fewer rounds of communication and much lower computational cost, especially because it does not require cryptographic operations.

5 A parallelized protocol for large scale translation

When transferring a large amount of data, a large proportion of the mappings are involved in the translation process. For example, if a query is to retrieve all diseases diagnosed in a hospital within a month, most diseases in the domain are returned. Answer privacy is still required as it may breach the privacy of minorities if a rare disease (associated to certain patients) is included in the result. At the same time, we still need to enforce mapping privacy to protect the translators' rights. Another issue is that, with a large number of mappings, each translator takes more time. Since translators work in a serial fashion, it takes long for the client to receive the results and most translators stay idle during the process, waiting for the entire mapping request to be received by the previous peer. In this section, we develop a parallelized privacy-preserving translation protocol (denoted by IMP, Item Mapping with Parallelization) **Nikos: explain what IMP means!** that addresses these issues.

We first explain the feasibility of parallelization. Without the presence of mapping request, a translator normally remains idle, as it is not able to determine what data should be translated. To overcome this difficulty, we propose a scheme, where the translator prepares all mapping entries for *all* values in the domain unconditionally. This ensures that the necessary mappings in the translation have been precomputed and they are ready to use, as soon as the mapping request is received. On the other hand, additional cost is required in preparing mapping entries for values that do not appear in the mapping request. Fortunately, the size of the mapping request can be determined by the server before any translation; thus, the server can decide whether the system should switch to the parallelized protocol or use the simple protocol. We now focus on the case of *full-domain translation* and discuss its parallelization.

5.1 Mapping representation: Index mapping

The current representation of mapping ($x \to y$) is *content-dependent*. In order to achieve composition of mappings, we have to match the right part of the first mapping to the left part of the second mapping; e.g., given $x \to y$ and $y \to z$, we have $x \to z$. From a privacy perspective, the above matching mechanism is not allowed because one can then tell which value is involved in the translation; thus, a more sophisticated protocol is required. In this section, we present a novel representation of mappings, called *index mapping*, which enables such a protocol.

The mappings of a translator T_i is a function that maps a value from a domain D_i to another domain D_{i-1}. The mappings of the next translator in the path T_{i-1} convert D_{i-1} to D_{i-2}. Note that the neighboring peers along the path share the same domain in the mappings as an interface to communicate. Each peer agrees on the same order (say alphabetical order) in each domain and represents each value in the domain as its index. The mapping function then operates at the index level. For example, suppose the domain D_i is ordered as {AIDS, heart disease, lung cancer}, the domain D_{i-1} is ordered as {d01, d02, d03}, and the original mapping function is {AIDS $\to$ d02, heart disease $\to$ d03, lung cancer $\to$ d01}. The index mapping is then $\{1 \to 2, 2 \to 3, 3 \to 1\}$. Index mapping only captures the mapping structure, without considering the semantics of values. Thus, one can compute the composition of index mappings without knowing the semantics of the values. However, in our case, translators also know the (ordered) domains of other translators. So, the index mappings cannot be revealed directly in our protocol. We adopt a token-based translation to hide them.

With index mapping, if one would like to translate the first value in D_i (AIDS), he could construct a size-3 vector with a token t in the first element while leaving the other two elements random (t, r_1, r_2)

where $t \neq r_1, r_2$. The index mapping then swaps the position of the values accordingly and returns (r_2, t, r_1). By finding the position of t, we know that it is mapped to the 2nd value of D_{i-1} (d02). In this way, the translator has performed the translation task without learning the value to be translated. Note that, the output and the input of the index mapping are in the same format, meaning that we can repeatedly apply different mappings to compute a composition of mappings. In fact, each mapping can be regarded as a permutation of the values in the set.

With index mappings, the server generates a random unique token for each value in the domain as initial encryptions to the values. This forms a token set V_{t+1} that represents the values in the context of the server. V_{t+1} is sent to translators for translation. Consider a serial translation. Each translator shuffles the tokens according to his index mapping and sends the shuffled tokens to the next translator. As the tokens are random, the shuffled tokens after applying an index mapping will appear random to translators; this protects the privacy of translators. At the same time the server sends V_{t+1}; i.e, the values in the query result are replaced by the corresponding tokens. The tokenized query result is sent to the client. So, by looking up the position of a token in the query result, the client is able to recover the translated value in his own context. In this process, the client is not able to observe the mappings of any translators, even when there is one translator only. Figure 3 shows an example illustrating the entire translation procedure. The server computes the query and the result contains 1 tuple only - ‘Bob’. Instead of sending to the client the result in the server’s context (‘Bob’), the server sends the tokens representing the result (‘0’). Note that ‘Bob’ is the 2nd value in the server’s domain, and it should be mapped to the 1st value in the client’s domain. The translator permutes the set of tokens and the token representing ‘Bob’ (‘0’) is put to the first slot in the tokens. The client, by identifying the positions of tokens in the query result, recovers the query result in its context.

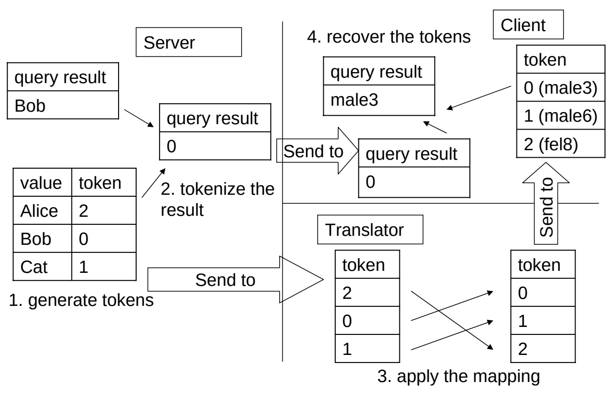

Figure 3: An example illustrating the procedure of the protocol.

Note that in order to ensure the correctness of the protocol, the tokens for different values in the domain must be different. On the other hand, the domain of tokens does not affect the correctness and the security of the protocol. For a domain D, we need to use $|D|$ tokens. A simple token domain is $[1, |D|]$; thus, randomly generating a set $|D|$ of unique tokens in the domain of $[1, |D|]$ is the same as generating a random permutation of $[1, |D|]$, which can be done in $O(|D|)$ time.

5.2 Parallelized permutation composition

Recall that an index mapping can be viewed as a permutation. A permutation can be represented by a matrix[10]; e.g., index mapping $\{1 \to 2, 2 \to 3, 3 \to 1\}$ can be modeled by:

$$M = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

[10]We remark that although we present our approach using a matrix representation, the actual content form of messages in the communication is permutation.

Similarly, we can represent the set of tokens as a row matrix so that $(t_1, t_2, t_3) \cdot M = (t_2, t_3, t_1)$. If the path from the client C to the server S is $C, T_1, T_2, ...T_t, S$, each translator T_i has an index mapping, which is represented by M_i. The initial tokens generated are also a permutation, so they can also be represented by a matrix M_{t+1}. The goal is to compute $\Pi_{i=1}^{t+1} M_i$, i.e., $M_{t+1}M_t...M_1$. By using divide and conquer, we can divide the computation into two halves: $\Pi_{i=1}^{\frac{t+1}{2}} M_i$ and $\Pi_{i=\frac{t+1}{2}+1}^{t+1} M_i$. Similarly, we recursively divide the computation until there are only two parties involved. This reduces the number of rounds from $O(t)$ to $O(\log(t))$.

In the first round, every two consecutive peers T_{i-1} and T_i form a group to compute M_iM_{i-1}. One of the two peers must contribute his mappings to the other peer for the computation. For example, T_i should send his mappings M_i to T_{i-1}. However, this violates the privacy of T_i. To protect M_i, T_i generates a random permutation R_i (represented as a matrix too). T_i's mapping is applied on the random mapping, i.e., $M_i' = M_iR_i$ (if T_{i-1} is sending his mappings, $R_{i-1}M_{i-1}$ is used). M_i' is sent to T_{i-1} to compute $M_{i+1}M_i'$. In order to maintain the correctness of the computation, R_i must be eliminated in the final result. T_i sends the inverse of the random noise R_i^{-1} to T_{i-1} and T_{i-1} computes $M_{i-1}' = R_i^{-1}M_{i-1}$ as his input to the computation procedure (if T_{i-1} is sending his mappings, the inverse is sent to T_{i-2} and T_{i-2} computes $M_{i-1}R_{i-1}^{-1}$). In the boundary case (e.g., T_1), the inverse is sent to the client. This is to ensure the correctness of the protocol when we combine different parts together, i.e., $M_i'M_{i-1}' = M_iR_iR_i^{-1}M_{i-1} = M_iM_{i-1}$. Note that, for each group of peers T_{i-1}, T_i, it must be the peer on the same side (either left - smaller index or right - larger index) to send the mappings to the other peer. So, alternate peers will be generating a random permutation.

Now, we do not need to consider the peers who have contributed their mappings to other peers. So, half of the peers remain. Another round is carried out similarly. Every two consecutive of the remaining peers form a group but now one of the peers just contributes his composite mappings to another peer without generating the random permutation again. This is because the composite mappings already contain a random permutation generated by the partner in the first round. For example, T_2 receives M_1R_1 from T_1 and R_3^{-1} from T_3. T_2 combines everything together and sends $R_3^{-1}M_2M_1R_1$ to T_4. Note that T_4 does not know R_3 and R_1 and the resulting permutation looks random to T_4. The same procedure is applied at later rounds. Figure 4 shows an example illustrating the computation of the composite mappings with 7 translators.

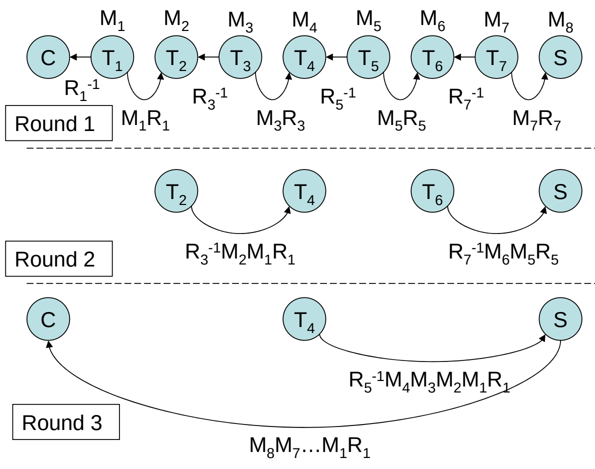

Figure 4: Parallelized Permutation Composition.

5.3 Security proof

Theorem 4 *The IMP protocol presented in this section satisfies k-protection for any $k \leq n$.*

Proof: Since no query request is sent to translators, the IMP protocol enforces n-protection for the full domain size n; i.e., the probability that each value is in the answer set is $\frac{1}{n}$. Trivially, IMP achieves k-protection for any $k \leq n$ as well.

Theorem 5 *The IMP protocol enforces mapping privacy.*

Proof: The intermediate mappings are hidden from the client. The client only knows the final mapping results from the server to the client. However, without knowledge of the value-token table on the server, the client cannot utilize these final results; therefore, the protocol enforces mapping privacy.

6 A Collusion-resistant scheme

Our previous schemes cannot guard against collusions where the parties in the colluding group share the messages they obtained in the protocol. For example, in Figure 4, translators T_2 and T_4 can share their knowledge (M_3R_3 and R_3^{-1}) and they can recover the mappings of T_3. In this section, we perform an analysis on the collusion problem and derive an efficient and collusion-resistant solution (denoted by PC, Privacy preserving with Collusion) **Nikos: explain what PC means!**. First, we study the potential risks of colluding scenarios in our protocols.

1. Answer privacy: In our proposed protocols, the server sends the necessary hints (i.e., answer key) to the client directly. Thus, translators cannot breach the privacy of the query result without the client or the server getting involved in the collusion. This is indeed the best that we can achieve; if either the server or the client is involved in the collusion, then the entire query result will be revealed to colluding parties regardless what translation protocol we use.

2. Mapping privacy: In our proposed protocols, the mappings of a translator can be recovered with a minimum of two colluding parties. We aim at raising the resistance of the protocol so that even when certain parties collude together, the mappings of a particular peer (not in the colluding group) cannot be recovered. Note that it is not possible to protect one's mappings if all other parties collude.[11] In this paper, we target the case of at least two parties being not in the colluding group.

To protect one's private mappings, we need to hide them by adding some secret parameters (like encryption with a key). On the other hand, the 'decryption key' has to be sent to other parties in order to cancel out the secret key in the final result. Since any other party may be in the colluding group, we break the decryption key into shares and each share is sent to a different party. Thus, each other party obtains a share of the decryption key. No party can recover the decryption key without acquiring all the shares. In addition, each share should be of $O(n)$ size or otherwise there is a leak of statistical information ($O(2^n)$ possible mappings but $O(2^k)$ key space for a size-k key). Each party needs to generate the shares of the decryption key and it also receives the shares of other decryption keys from other parties. The processing cost for each party is $O(mn)$: a significantly increase compared to the $O(n)$ cost in the previous protocols. Thus, the protocol having perfect security is very expensive.

Our goal is to design a more efficient protocol. In particular, we aim at limiting the processing cost to $O(n)$ while keeping adequate protection for guarding against collusions. In other words, we want the secret parameters and shares of decryption key to be of $O(1)$ size and to be aggregated efficiently (with linear cost). In summary, the protocol goals are as follows:

1. The colluding group cannot derive the query result unless the client or the server is involved in the group.

2. The colluding group cannot derive the mappings of a party not in the group as long as there are at least two parties not in the group.

3. The protocol incurs a low overhead.

[11]The client obtains the mappings from the server's context to the client's context by matching the query results under the two schemas when the client and the server collude together. This mapping is equivalent to the composition of mappings of translators. Since all parties except one collude together, we can solve the mappings of the remaining party.

6.1 Guarding against collusion via pseudo-permutation

This section describes a protocol satisfying the above requirements. One technical challenge is that we need to cancel out the secret parameters added by each translator. Encryption is usually not commutative. This requires a correct ordering in encrypting or decrypting the mappings using the partial keys and makes the protocol complicated and expensive. To solve this problem, we use a *pseudo-permutation* approach, which resembles commutative encryption but it is simple and cost-friendly.

Definition 2 (Pseudo-permutation) *Consider a vector v (tokens) that is a permutation of n numbers (*a_1*,* $a_2, \ldots, a_n$*). A pseudo-permutation f takes a parameter r that is randomly sampled in* $[1, n]$ *and gives* $f(r, v)$ *= (*$ra_1 \mod n+1$*,* $ra_2 \mod n+1, \ldots, ra_n \mod n+1$*) where r and* $n+1$ *are relatively prime.*

Note that we require r and $n+1$ are relatively prime. This is true if we consider $n+1$ is a prime.

For example, consider a set of tokens v $(1, 2, 3, 4)$ $(n + 1 = 5)$ and $r = 3$. The resulting set of tokens is $(3, 1, 4, 2)$ (e.g., 2nd position: $(2 \cdot 3) \mod 5 = 1$). To recover the original set of tokens, we can multiply the set of tokens with the modular multiplicative inverse of r (we can compute the modular multiplicative inverse using the Extended Euclidean algorithm). In our example, using $r^{-1} = 2$, we get the tokens $(1, 2, 3, 4)$ (e.g., 3rd position: $(4 \cdot 2) \mod (5) = 3$). The constraint that r and $n + 1$ are relatively prime ensures that the modular multiplicative inverse of r exists and thus makes the pseudo-permutation 'decryptable'.

Pseudo-permutation is like a random permutation (Lemma 1) but has a smaller key size; it is more efficient, but offers a weaker protection. If we use a matrix representation, a pseudo-permutation is equivalent to $\mathcal{R} = \begin{pmatrix} r & 0 & 0 \\ 0 & r & 0 \\ 0 & 0 & r \end{pmatrix} = rI$, where r is a numeric value and I is the identity matrix. Pseudo-permutation is commutative to other permutations. Given any pseudo-permutations $\mathcal{R}_1$, $\mathcal{R}_2$ and a permutation P, we have $\mathcal{R}_1 P = P\mathcal{R}_1$ and $\mathcal{R}_1\mathcal{R}_2 = \mathcal{R}_2\mathcal{R}_1$. Besides, composition of two pseudo-permutations can be done by a simple multiplication. Given any pseudo-permutation $\mathcal{R}_1 = r_1 I$, $\mathcal{R}_2 = r_2 I$, $\mathcal{R}_1\mathcal{R}_2 = (r_1 r_2)I$. With pseudo-permutation, we can construct the protocol for guarding against collusion as follows:

(1) Preparation phase. The server S computes the query result and generates noise to form the mapping request. S also generates a token set V, that is a permutation of $[1, n]$, to represent the values in the mapping request. The query answer is re-written using the tokens and sent to the client C. At the same time, S informs all other peers about the size of the mapping request.

(2) Anti-collusion parameter generation phase. Each translator T_i generates a random pseudo-permutation by generating t pairs r_{ij} and r_{ij}^{-1} with respect to the modulo $n + 1$, where n is the number of values in the mapping request. Each r_{ij} is sent to a different peer while T_i keeps its secret parameter $\lambda_i = (\prod_{i=0}^{t+1} r_{ij}^{-1}) \mod n + 1$ (we denote the client as T_0 and the server as T_{t+1} here for easier discussion). As a result, each peer (including the server, the client and translators) denoted by T_j, will receive a set of random numbers r_{ij}, which is a share of the decryption key of T_i. The peer aggregates them into one value by multiplying them together. If the peer is a translator, he also multiplies his secret parameter λ_i as generated above. We denote the resulting value by $\mathcal{R}_X$ where X denotes the peer having the value.

(3) Translation phase. S sends to the neighboring translator T_t, $\mathcal{R}_S V$ and the mapping request. Let T_i be a translator, which receives a set of tokens and the mapping request. Let V' be the set of tokens received. After T_i applies his mapping on the mapping request, he randomly permutes the values in the mapping request. Let the permutation be P_i, T_i also applies the same permutation P_i on the set of tokens, followed by $\mathcal{R}_{T_i}$, i.e., T_i sends to the next translator $\mathcal{R}_{T_i} P_i V'$.

(4) Answer recovery phase. The client C obtains a set of translated values: the mapping request and a set of tokens V'. C replaces the query answer, that is written using tokens, according to the position of the tokens.

Correctness. The correctness of the protocol relies on whether the permutation of the tokens is the same as the permutation of the values in the mapping request. Given the initial set of tokens V generated by S, our goal is to allow C to observe $(\prod_{i=1}^{t} P_i)V$. Note that the set of tokens that the client received in the protocol is $(\prod_{i=1}^{t} \mathcal{R}_{T_i} P_i)V = (\prod_{i=1}^{t} \mathcal{R}_{T_i})(\prod_{i=1}^{t} P_i)V$ (due to the commutative property of pseudo-permutation). Since we generate each component of pseudo-permutation in pairs of multiplicative inverse, we have $\prod_{i=1}^{t} \mathcal{R}_{T_i} = I$. The client obtains the set of tokens that go through the same permutation as the mapping request and thus he can recover the query answer correctly.

Example. Assume there is one translator T only. Suppose the query answer is (m_1, m_2, m_1) and the mapping request after random permutation at S is $(m_\alpha, m_2, m_1, m_\beta)$. S also generates a set of tokens, a permutation of 1 to 4 to represent the values. Assuming the set of tokens is $(2, 1, 3, 4)$, S re-writes the query answer using the tokens and sends to C the query answer $(3, 1, 3)$. Then, we go to the second phase to generate the random parameters. Suppose T generates two pairs of multiplicative inverse (2, 3) and (4, 4) (note that $(2 \cdot 3) \mod 5 = 1$ and $(4 \cdot 4) \mod 5 = 1$). T sends to S a number in the first pair, say 2, and sends to C a number in the second pair, 4. T keeps $3 \cdot 4 \mod 5 = 2$ for eliminating the pseudo-permutations. Then, we go to the translation phase. S first applies his pseudo-permutation $\mathcal{R}_S = 2I$ on the set of tokens. The resulting set of tokens $(4, 2, 1, 3)$ is then sent to T. Assume the mapping of T is $m_x \rightarrow m'_x$. After translation on the mapping request, T applies the same random permutation on the mapping request and the tokens. Assume that the mapping request and the tokens become $(m'_2, m'_\beta, m'_\alpha, m'_1)$ and $(2, 3, 4, 1)$, respectively. T then applies his pseudo-permutation $\mathcal{R}_T = 2I$ and the tokens become $(4, 1, 3, 2)$. The translated mapping request and the tokens are sent to C and C applies his pseudo-permutation $\mathcal{R}_C = 4I$ on the tokens. The tokens become $(1, 4, 2, 3)$. Finally, C recovers the query answer. Recall that the query answer is expressed using tokens $(3, 1, 3)$. C replaces each token by the value in the mapping request that has the same position of the token, i.e., (m'_1, m'_2, m'_1).

6.2 Security proof and cost analysis

Theorem 6 (Answer privacy) *The PC protocol satisfies k-protection.*

Proof: The necessary hint to recover the query answer (the answer key expressed using tokens) is shared between the server and the client only. Thus, translators are not able to observe the query answer with the presence of tokens. Translators can observe hints of the query answer from the mapping request, but the fake values in the mapping request enforce k-protection.

Theorem 7 (Mapping privacy) *Assuming that $n + 1$ is prime, the PC protocol enforces mapping privacy on any translator T, unless all parties except T collude together.*

Lemma 1 *Let n be a prime number. Let r be a random number chosen from $[1, n-1]$ such that $gcd(r, n) = 1$. We define $f_r : [0, n-1] \rightarrow [0, n-1]$ via $f_r(a) = ra \mod n$ for any $a \in [0, n-1]$. Then, f_r is a (pseudo) random permutation on $[0, n-1]$.*

Proof: First, we show that f_r is a permutation. Suppose that there exists a_1, a_2 such that $ra_1 \mod n = ra_2 \mod n$. Then, as $gcd(r, n) = 1$, there exists c such that $cr \mod n = 1$. Therefore, $a_1 = cra_1 \mod n = cra_2 \mod n = a_2$.

Now we show f_r is pseudo random. Let b be chosen randomly from $[0, n-1]$. If r is a random number, then $\Pr[f_r(a) = b] = \Pr[ra \mod n = b] = \Pr[a = (r^{-1} \mod n)b] = \frac{1}{n}$. We note that the above argument is true only if r is random. When n is prime, this is true, as $gcd(r, n) = 1$ for any $r \in [1, n-1]$.

Proof: (Proof of Theorem 7) The worst case is when neighboring peers of a translator T collude together. In such a case, the colluding group observes the mapping request and the tokens before and after T's translation. However, since T applies a random permutation P on the mapping request, the colluding group cannot recover the exact mapping of T unless the group recovers P. Let V be the set of tokens that T receives and $\mathcal{R}_T$ be the pseudo-permutation applied by T, provided that $n + 1$ is prime (Lemma 1). The colluding group observes V and $V' = \mathcal{R}_T P V$. With V and V' known there are $|V|$ equations with $|V| + 1$ unknowns; we can solve the above system but there are multiple solutions to $\mathcal{R}_T$ and P. Since the components of $\mathcal{R}_T$ are pairs of modular multiplicative inverses and each pair is owned by two peers, the colluding group can also invite others to join the group in order to obtain the necessary information to recover $\mathcal{R}_T$. However, $\mathcal{R}_T$ contains shares of every other peer. Therefore, unless all parties except T are in the colluding group, $\mathcal{R}_T$ cannot be recovered.

Cost analysis. Compared to the protocols we discussed in previous sections, our protocol here incurs additional cost for the anti-collusion measures. Each component of pseudo-permutation is a pair of modular multiplicative inverses. Thus, the parameters are all of $O(1)$ size. Each translator generates $O(t)$ pairs and receives $O(t)$ pairs from other parties. The communication cost and the computational cost to aggregate them into a single pseudo-permutation are both $O(t)$. In the translation phase, each translator translates the mapping request (at $O(kn)$ time) and applies permutations on the mapping request and the tokens (at $O(kn)$ time). Thus, the overall cost at a translator is $O(kn + t)$, where k is the privacy parameter and n is the number of distinct values in the query answer. Note that since t is usually smaller than kn, the complexity of the protocol here is basically the same as the simple protocol we described in Section 4.

7 Experimental Study

In this section, we evaluate the performance of the proposed solutions. Our implementation of the PPP protocol using oblivious transfer is denoted by POT (see Section 3). POT-opt denotes the same protocol, after applying all the optimizations proposed in Section 3.2. The lightweight protocol proposed in Section 4 that does not use cryptographic operations is denoted by PD. The parallelized protocol using index mapping (Section 5.2) is denoted by IMP. Finally, PC denotes the collusion-resistant protocol of Section 6.

We compared our methods with two approaches. (1) The PPP protocol which is implemented using Pohlig-Hellman commutative encryption [21]. (2) NP is a basic query processing algorithm which does not consider privacy; in NP, the server sends the query result directly to client and also sends the distinct values to the client through the path of the translator peers, which translate them on the way to the client. The cost of NP is a lower bound for any privacy preserving protocol. Table 1 summarizes the security level of the evaluated privacy-preserving protocols.

All algorithms are implemented in C++. Each peer is executed on an individual Intel Core2 Duo 2.83GHz machine with 3.2GB RAM, running Windows. The peers are connected through a 1Gbps LAN. In the experiments of Sections 7.1–7.3 we use synthetic data. Real data are used in Section 7.4. The synthetic dataset contains one attribute, which is the attribute queried by the client. Each tuple is assigned a random unique value on this attribute. So, the number of tuples equals the domain size of the attribute, denoted by $|D|$. We do not include duplicate values in the dataset, because the performance of the protocols mainly depends on the number of distinct values in the result. The mappings of each peer are randomly generated. In the experiments, we measure the time (time spent since client issues the query until he receives the plain results) and the communication cost of a random range query. Table 2 summarizes the privacy, network, data, and query parameters for the synthetic data experiments, and shows the range and default value of each parameter. Note that although we evaluate the costs of all protocols together, their security level strengths vary. Table 1 shows the different security levels achieved by each protocol. The detailed security proofs are derived from Theorems 1 to 4.

Note that PPP and POT (POT-opt) protocols do not strictly provide mapping privacy. Translators will not see each other's mappings but the client can. In the extreme case, the client can request and store all the mappings eventually in PPP and POT protocols and remove the translators from the chain at future queries.

	without collusion		**with collusion**	
Protocol	**k-protection**	**mapping** privacy	**k-protection**	**mapping** privacy
PPP	X	reveal to client only	X	reveal to client only
POT POT-opt	✓	reveal to client only	✓	reveal to client only
PD	✓	✓	✓	X
IMP	✓ (for all $k \leq n$)	✓	✓	X
PC	✓	✓	✓	✓

Table 1: Privacy protection strength of algorithms.

Parameter	Values
Privacy parameter k	2, 4, **6**, 8, 10 (for k-protection only)
Domain size $\|D\|$	2000, 6000, **10000**, 20000, 40000, 60000, 80000
Number of peers n_p	3, 5, **9**, 17, 33
Query selectivity s (in %)	1, 5, **10**, 15, 20, 25, 30

Table 2: Parameters used in experiments on synthetic data. Default values in bold font.

7.1 k-protection

In the first experiment (Figure 5), we examine the performance of algorithms w.r.t. k-protection. Note k does not exceed 10 because the query selectivity in this experiment is 10%. IMP operates on the entire domain. Noise in mapping requests is not required and thus the performance of the protocol is not affected by k. The query times and communication costs of other privacy-preserving protocols (PPP, POT, PC, PD) increase with k. Our lightweight protocols (PC, PD, IMP) have lower query times and communication costs than PPP and POT because they do not use expensive cryptographic operations. The communication costs of cryptography-based protocols are also higher. This is because each plain data item has a small size (4 bytes in our experiment) while the encrypted data are much larger (up to 1024 bits (128 bytes) for PPP, POT, POT-opt). Observe also that k has little effect on the message cost of POT approaches. This is because our implementation of OT has an optimized size for encrypted mapping, where the encrypted mapping has the same size as the plain text (line 4, Algorithm 1). Thus, the cost of sending encrypted mapping entries of fake items is relatively low compared to other messages like OT requests. Note that a bigger k only affects the number of mapping entries to be transferred in POT-opt because we have pre-computed the encrypted mapping.

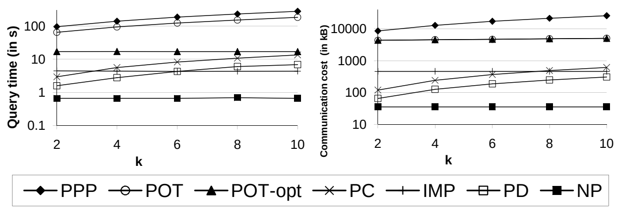

Figure 5: Query time and communication cost vs k on synthetic data.

7.2 Scalability

Figures 6 and 7 compare query times and communication costs, showing that all algorithms are scalable to both domain size and number of peers, respectively. PPP has the highest cost; the method becomes impractical for values of $|D|$ and n_p that are not trivially small. PC, PD, IMP have the lowest costs among all privacy-preserving protocols. Compared to NP, all methods are significantly expensive; the major overhead is due to the additional work on translation. Since we set $k = 6$, the number of values to be translated is 6 times more than in NP. For example, the overhead introduced by PD over NP at $n_p = 33$ has a factor close to k ($\frac{15.7}{2.6} = 6.04$). This indicates that PD incurs the minimal overhead to ensure k-protection. In general, the overhead of our methods over NP is bearable given the privacy protection that they provide. Compared to PPP, all our algorithms have a significant cost improvement (besides the better privacy protection they offer). The cost of IMP increases slower with n_p compared to the other methods. At $n_p = 33$, IMP takes 5.9s while NP takes 2.6s. This shows that parallelization can effectively control the cost when the query result has to travel along a long path, which is likely to happen in a large peer-to-peer network.

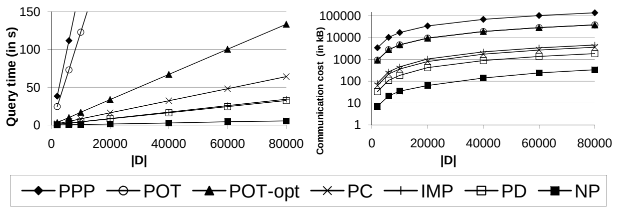

Figure 6: Query time and communication cost vs $|D|$ on synthetic data.

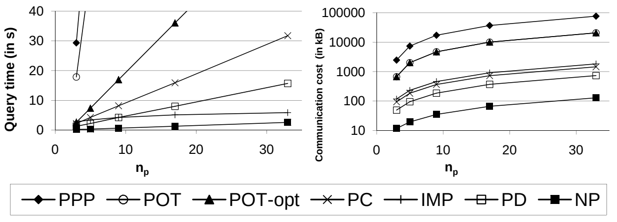

Figure 7: Query time and communication cost vs n_p on synthetic data.

7.3 Varying selectivity

Figure 8 shows the query times and communication costs of the algorithms with varying query selectivity. When s reaches 20, the query times of PC and PD become stable, because all values in the domain are included in the mapping requests when $s > \frac{100}{k}$. Further increase in s does not increase the size of mapping request and thus the query time is stabilized. In contrast, PPP and POT approaches show a stable increase with varying s; the major overhead in these protocols are at clients obtaining required mappings from translators using cryptographic operations. The number of the required mapping entries increases with s. Thus our lightweight protocols are especially beneficial when the query answer is large.

7.4 Experiments on real data

We also evaluated the performance of algorithms (PPP, POT-opt, PC, IMP, PD, NP) on two real datasets: (i) CENSUS (downloadable from 'http://www.ipums.org'), containing personal information of 50K individuals with 8 dimensions (average domain size 28.1) (ii) ADULT [2], containing personal information of 32,561 individuals with 15 dimensions (average domain size 1476.4). We generate for each peer a random mapping that maps each value to a different integer. Using the default values for the parameters, as shown in Table 2, we issue 20 random queries in the form of 'SELECT A1 FROM DATASET WHERE A2 IN (QS)', where A1 and A2 are two randomly chosen attributes in the dataset (A1, A2 can be the same) and QS is a set of values in the domain of A2 and QS contains around 10% elements in the domain. In addition to query time (query processing + answer translation), we also measure the query processing time alone at the server, denoted by QP. Table 3 shows the results.

Dataset	PPP	POT -opt	PC	IMP	PD	NP	QP
CENSUS	4.23	2.00	0.80	1.17	0.77	0.59	0.57
ADULT	23.30	3.43	1.90	1.97	1.33	0.68	0.58

Table 3: Query time (in s) on real dataset.

The results show that our proposed algorithms are much more efficient than PPP. Besides, our proposed algorithms have a lower query time overhead over NP than what we observe in the experiments on synthetic data. This is because our synthetic data do not have duplicate values; with more duplicate values in the dataset, the query processing time (QP) increases (as the distinct values need to be extracted and translated); in this experiment, query processing becomes a major component of the entire cost and the overhead in

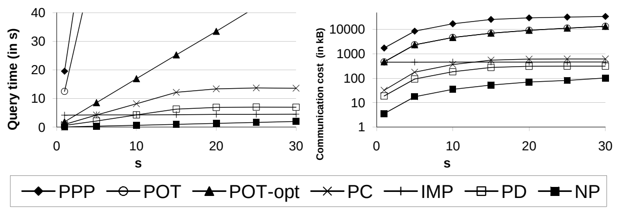

Figure 8: Query time and communication cost vs s on synthetic data.

query translation for protecting privacy becomes relatively cheaper. Thus, in realistic cases where the attribute domain is much smaller than the number of tuples in the database and duplicates are expected to appear in the query result, our protocols become even more practical, as they only incur a small overhead on top of query processing.

7.5 Simulation of large P2P network

In this experiment, we simulate large P2P network on an Intel Core2 Duo 2.83GHz machine with 2GB RAM, running Windows. We vary the number of translators from 1 to $10,000$ (i.e. n_p varies from 3 to 10,002). We use the default values for other parameters as shown in Table 2 and test all algorithms except IMP in this setting, as the parallelism in IMP cannot be realized in the simulation. Figure 9 shows the query times of different algorithms varying n_p.

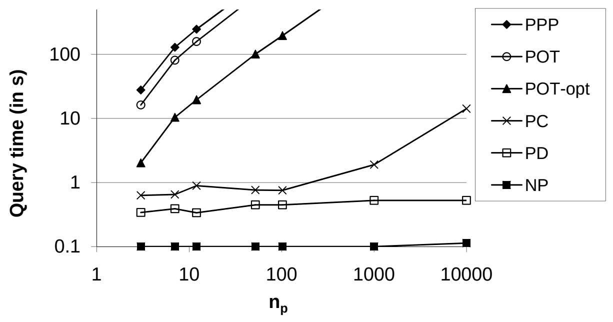

Figure 9: Simulated query time vs n_p on synthetic data.

The result shows that cryptographic approaches (PPP, POT and POT-opt) are more expensive than non-cryptographic approaches (PC and PD). The cryptographic operations are very expensive. By avoiding them, PC and PD reduce the cost for privacy by orders of magnitude. On the other hand, PC, PD and NP show a steady cost for small n_p; this is because the communication cost is not realized in our simulation. Translation has a very low processing cost compared to query processing (and noise generation for PC, PD). Thus, the measured query time is almost constant unless n_p reaches a large value, which renders the translation cost significant. Compared to the query time measured in a real network setting in Figure 7, the processing cost we measured here is very low. So, we expect that the communication cost will be the dominating factor for non-cryptographic algorithms on large-scale networks.

7.6 Mapping privacy leakage

In this experiment, we test on how many mappings are revealed to client as time goes. Note that only PPP, POT, POT-opt may leak the mappings to the client (see Table 1) and they will leak the same amount of mappings to client as all algorithms use the same framework to generate the fake answers. We vary the values of k and s and measure how many mappings are revealed to client after x queries for x = 1 to 100. The experiment is done on the default synthetic dataset (with 10k domain size). Figure 10 shows the experiment result.

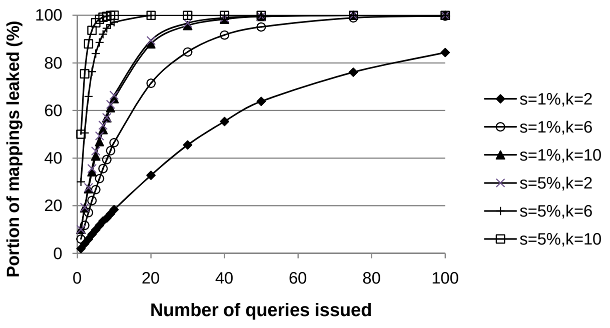

Figure 10: Portion of mappings revealed to client as queries are issued.

The result shows that the non-mapping-privacy-preserving algorithms can easily leak most of the mappings with a relatively small queries. For example, if $k = 6$, $s = 1\%$, 47% of mappings are leaked to the clients in 10 queries while 85% of mappings are leaked in 30 queries. Note in each leakage that, the client is able to obtain the mappings of each translator on values in the query answer and the fake answers. If there are 7 translators (9 peers in total with client and server), the client can take the translator jobs that are destined to the 7 translators and thus reduce the profitability of the original owners of the mappings. This discourages peers from setting up accurate mappings between them, which usually requires much human efforts and is thus expensive.

7.7 Mapping privacy leakage in collusion

In this experiment, we test on the security impact on mapping privacy when two peers collude. We assume a worst case scenario in 2-party collusion. The two neighbor peers of a victim peer (which is a translator) are colluding, i.e., the two malicious peers are exchanging the messages they observed in the algorithms and they will try to derive the mappings of the victim peer. Note that PPP, POT, POT-opt and PD have the same protection strength against collusion and we will use 'G1' to represent this group of algorithms. We use the default settings as shown in Table 2. Figure 11 shows the portion of mappings leaked to the colluding parties against number of queries issued.

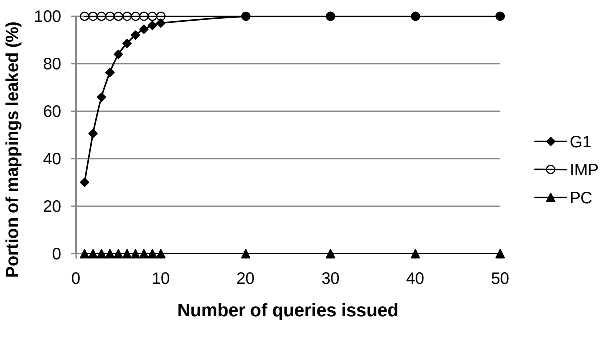

Figure 11: Portion of mappings revealed to client when two peers are colluding.

The result shows that all algorithms except PC have a very poor protection against collusion. When two peers collude, they can easily observe the mappings of the victim in a few queries. For IMP, the entire mapping is revealed in only 1 query because the translation is done on the entire domain. PC, on the other hand, can protect mapping privacy even in a collusion scenario, as it is designed to be.

8 Related Work

Peer Database Management Systems (PDMS) that leverage peer-to-peer techniques to manage dynamic and heterogeneous data have received much attention in the recent years. Examples include [19, 27, 23, 8]. However, none of these works considered the privacy issues in a PDMS as discussed in this paper. The problem of protecting query answers in PDMS was first investigated in [7], which defined the privacy notion k-protection for this problem and devised a solution for the semi-honest model [10] based on commutative encryption. However, to our knowledge, no secure commutative encryption scheme in practice can be instantiated in this scheme to satisfy k-protection.

There are several works (e.g., [1, 28, 11]) on privacy issues in data mining. Most of them employ tools from Secure Multi party Computation (SMC) [10]. Generally speaking, each party i in SMC has its own input x_i and the parties want to cooperate to calculate $f(x_1, \ldots, x_n)$ without any party i learning anything beyond $f(x_1, \ldots, x_n)$ and its own input x_i. Oblivious transfer [22] is a dedicated SMC protocol (specifically, secure two-party computation): there is a sender with message $M_1, \ldots, M_N$ and a receiver with an index $\sigma \in \{1, \ldots, N\}$; these two parties engage in an OT protocol where the receiver learns M_σ only and the sender learns nothing about the receiver's choice σ. SMC protocols can be built based on OT, but they are still very expensive. The reason is that SMC works only for some certain NP-complete problems. Therefore, when we apply SMC to a database problem P, we have to reduce P into a certain NP-complete problem by embedding P into a circuit. This reduction method is generic but highly inefficiently involving a large number of OT protocols. Therefore, even when each OT operation is cheap, the resulting SMC becomes very expensive. We should also note that although the functionality of OT is similar to that of Private Information Retrieval (PIR) [6, 9, 20], PIR does not protect the mappings of the translators. For example, PIR allows the client to see the whole mapping table in plaintext (undesired in our problem), while in OT, the client learns M_σ only.

In *privacy-preserving data publishing*, a data owner owns a database and would like to make it public. The data may contain sensitive information, so data sanitization is required before publication. Several privacy notions are developed, e.g., k-anonymity [25], l-diversity [16], and t-closeness [14]. For example, k-anonymity requires that an attacker cannot identify a sanitized tuple in the published data with a probability larger than $\frac{1}{k}$. Some of these privacy principles are also used in some other applications, e.g., k-anonymity is used in outsourced location-based services [31]. Unfortunately, most of these definitions are not practical [12, 30], as they consider attackers of limited knowledge, while it is hard to estimate the attacker's capability and knowledge in practice.

9 Conclusion

In this paper, we studied the problem of privacy-preserving querying between peers with different schemas in a PDMS. We first unveiled that the state-of-the-art solution (PPP [7]) is inappropriate if commutative encryption is used. As a fix, we adopted an oblivious transfer technique and provided optimizations to make it practically faster. Still, the framework of PPP cannot adequately address the issue of mapping privacy and it is computational expensive due to the heavy use of cryptographic operations. To address these issues, we developed two lightweight protocols: one of them adopts serial translation and it is suitable for translating small query answers; the other adopts parallel translation and it is suitable for large-scale translations. Finally, we also considered a stronger adversary model where peers may collude. Again, a lightweight protocol is devised to guard against such scenarios. We analyzed the privacy and cost of our proposed solutions and experimentally studied their performance on synthetic and real datasets. The results show that they are much more efficient than PPP despite of the fact that they offer better privacy protection.

References

[1] R. Agrawal and R. Srikant. Privacy-preserving data mining. *SIGMOD Rec.*, 29(2):439–450, 2000.

[2] A. Asuncion and D. Newman. UCI Machine Learning Repository, 2007.

[3] M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko. The one-more-RSA-inversion problems and the security of chaum's blind signature scheme. *Journal of Cryptology*, 16:185–215, 2003.

[4] A. Bonifati, E. Q. Chang, T. Ho, L. V. S. Lakshmanan, R. Pottinger, and Y. Chung. Schema mapping and query translation in heterogeneous P2P XML databases. *VLDB J.*, 19(2):231–256, 2010.

[5] J. Camenisch, G. Neven, and A. Shelat. Simulatable adaptive oblivious transfer. In *EUROCRYPT*, 2007.

[6] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. In *FOCS*, 1995.

[7] H. Elmeleegy, M. Ouzzani, A. Elmagarmid, and A. Abusalah. Preserving privacy and fairness in peer-to-peer data integration. In *SIGMOD*, 2010.

[8] E. Franconi, G. Kuper, A. Lopatenko, and I. Zaihrayeu. Queries and updates in the coDB peer to peer database system. In *VLDB*, 2004.

[9] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K.-L. Tan. Private queries in location based services: anonymizers are not necessary. In *SIGMOD*, 2008.

[10] O. Goldreich. *Foundations of Cryptography, Volume 2*. Cambridge University Press, 2004.

[11] M. Kantarcioglu and C. Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. *IEEE Trans. on Knowl. and Data Eng.*, 16(9), 2004.

[12] D. Kifer. Attacks on privacy and definetti's theorem. In *SIGMOD*, 2009.

[13] N. Koblitz. *A Course in Number Theory and Cryptography, 2nd ed.* Springer-Verlag, 1991.

[14] N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *ICDE*, 2007.

[15] R. Lipton. How to cheat at mental poker. In *Proceedings of the AMS Short Course in Cryptography*, 1981.

[16] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. In *ICDE*, 2006.

[17] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[18] National Institute of Health. Computational technology for effective health care: Immediate steps and strategic directions, 2009. http://www.nlm.nih.gov/pubs/reports/comptech_prepub.pdf.

[19] W. S. Ng, B. C. Ooi, K.-L. Tan, and A. Zhou. PeerDB: A P2P-based system for distributed data sharing. In *ICDE*, 2003.

[20] S. Papadopoulos, S. Bakiras, and D. Papadias. Nearest neighbor search with strong location privacy. In *VLDB*, 2010.

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

[22] M. O. Rabin. How to exchange secrets with oblivious transfer. Technical Report TR-81, Aiken Computation Laboratory, Harvard University, 1981.

[23] P. Rodríguez-Gianolli, A. Kementsietsidis, M. Garzetti, I. Kiringa, L. Jiang, M. Masud, R. J. Miller, and J. Mylopoulos. Data sharing in the hyperion peer database system. In *VLDB*, 2005.

[24] A. Shamir, R. L. Rivest, and L. M. Adleman. Mental poker. Technical Report LCS/TR-125, Massachusetts Institute of Technology, 1979.

[25] L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5), 2002.

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

[27] I. Tatarinov, Z. Ives, J. Madhavan, A. Halevy, D. Suciu, N. Dalvi, X. L. Dong, Y. Kadiyska, G. Miklau, and P. Mork. The piazza peer data management project. *SIGMOD Rec.*, 32(3):47–52, 2003.

[28] J. Vaidya and C. Clifton. Privacy preserving association rule mining in vertically partitioned data. In *KDD*, 2002.

[29] S. A. Weis. *New Foundations for Efficient Authentication, Commutative Cryptography, and Private Disjointness Testing*. PhD thesis, Massachusetts Institute of Technology, 2006.

[30] R. C.-W. Wong, A. W.-C. Fu, K. Wang, and J. Pei. Minimality attack in privacy preserving data publishing. In *VLDB*, 2007.

[31] Y. Yang, S. Papadopoulos, D. Papadias, and G. Kollios. Spatial outsourcing for location-based services. In *ICDE*, 2008.

[32] Y. Zhang, W.-K. Wong, S. Yiu, N. Mamoulis, and D. W. Cheung. Lightweight privacy-preserving peer-to-peer data integration. Technical Report TR-2011-12, University of Hong Kong, CS Dept., http://www.cs.hku.hk/research/techreps/document/TR-2011-12.pdf, 2011.

A Attack to Pohlig-Hellman Encryption

To understand the attack, we need to first describe the mechanism of Pohlig-Hellman encryption. The encryption and decryption keys are defined as (e, p) and (d, p), respectively; p is a large prime number. e is randomly chosen from $[1, p-2]$ and $ed \mod (p-1) = 1$. [12] Note that the multiplicative inverse of $e \mod (p-1)$ exists if and only if $\gcd(e, p-1) = 1$, i.e., e and $p-1$ are relatively prime (similar for d). Both keys are kept secret. Given a message $m < p$, the encrypted message is $E_e(m) = m^e \mod p$. Given a ciphertext c, the decrypted message is $D_d(c) = c^d \mod p$. The correctness of the encryption/decryption is ensured, so that $m^{ed} \mod p = m$ if $ed \mod (p-1) = 1$ [21].

The intuition of the attack to Pohlig-Hellman encryption works as follows. An integer x is said to be a *quadratic residue* (QR) modulo p if $\exists y \in \mathbb{Z}_p^*, x = y^2 \mod p$. All other values that are not quadratic residue are called quadratic non-residue (QNR) modulo p. In Theorem 8, we show that the plaintext and the corresponding ciphertext must be both QR or both QNR. This provides us additional information in identifying the original value of encrypted messages.

Theorem 8 *A message m is QR (modulo p) if and only if its ciphertext $c = E_e(m)$ in Pohlig-Hellman commutative encryption is QR (modulo p).*

Proof: If m is QR, we have $m = x^2 \mod p$ for some $x \in \mathbb{Z}_p^*$. Then, $c = m^e \mod p = (x^e)^2 \mod p$. Hence, c is QR as well.

The only if case can be proven similarly as $m = c^d \mod p$. We have $c = x^2 \mod p$ for some x if c is QR. Hence, $m = (x^d)^2 \mod p$. m is QR.

To test whether a message is QR, [13] proved that an integer x is QR modulo p if and only if $x^{\frac{p-1}{2}} \mod p = 1$. In the case of PPP, a translator observes a set of encrypted mappings M from the client and the domain of plaintexts D. $|D| = k|M|$ with k-protection. Let D_{QR} be the set of values in D that are QR, M_{QR} be the set of encrypted values in M that are QR. Since the encrypted value of QR must be QR, the certainty of a mapping in D_{QR} belonging to the result is $\frac{|M_{QR}|}{|D_{QR}|}$. Note that the distribution of QR and QNR may not be even, the certainty may be as high as 1 in an extreme case when $|M_{QR}| = |D_{QR}|$.

[12] To facilitate the commutative property, different parties have to use the same modulo p. On the other hand, e and d are kept secret for each party and are not allowed to share among multiple parties.

Obliviously, there is one (i.e., m'_1) equal to 1. Note that $\Pr[m'_1 \in R^T] = 1 > \frac{1}{2}$ has violated k-protection with $k = 2$.

Example. Consider a prime number $p = 5$. $\{1, 4\}$ are QR ($1 = 1^2 \mod 5 = 4^2 \mod 5$, $4 = 2^2 \mod 5 = 3^2 \mod 5$) while $\{2, 3\}$ are QNR. In general, given a prime p, $\frac{p-1}{2}$ integers in $\mathbb{Z}_p^*$ are QR and another $\frac{p-1}{2}$ in $\mathbb{Z}_p^*$ are QNR.

Suppose that the client C picks $(e_1 = 3, p = 5)$ and that the translator T picks $(e_2 = 5, p = 5)$ as the encryption keys of Pohlig-Hellman encryption (recall that $\gcd(e, p-1) = 1$, so e cannot be 2, 4). Assume '$x \to 2$' is the desired mapping by C and '$y \to 4$' is a fake mapping request added by the server S to confuse T (x, y are known values). In PPP, the two mappings are encrypted by T and are sent to C. C obtains '$x \to E_{e_2}(2) = 2$' and '$y \to E_{e_2}(4) = 4$' and sends back the double-encrypted mapping of x. T obtains $E_{e_1}(E_{e_2}(2)) = 3$. Note that the double-encrypted value 3 is QNR. 2 is QNR and 4 is QR. Without knowing the value of e_1, T can conclude that C is requesting the mapping $x \to 2$.

We note that the above attack is applicable to other exponential encryption schemes, like SRA, as well.

B The oblivious transfer protocol

To use OT, each translator T needs a (public key, private key) pair (pk, sk). The key can be generated by first generating two prime numbers p and q. Then, T computes $n = pq$ and also randomly chooses $e < n$ such that $\gcd(e, (p-1)(q-1)) = 1$ (so that there is a multiplicative inverse $e^{-1} \mod (p-1)(q-1)$). T computes d such that $ed \mod (p-1)(q-1) = 1$, publishes $pk = (e, n)$, and keeps $sk = (d, n)$ as secret. (This procedure is the same as RSA.) For each mapping $x \to y$, y is a possible data item to be retrieved by the client C while x is the index of the data. T encrypts his mappings using: $E(x \to y) = x \to c = H_2(x, H_1(x)^d \mod n) \oplus y$ where H_1 and H_2 are any cryptographic hash functions that are known to public, and $\oplus$ represents exclusive OR operation.

Assume that C has received the set of encrypted mappings from T. In the OT protocol, there are two rounds of communication. One is a retrieval request from C and one is the response from T which allows C decrypt the desired mapping without revealing the other mappings. When C requests for a data with an index x, it first randomly chooses $r < n$. He prepares an OT request for the data with $\sigma = r^e \cdot H_1(x) \mod n$ (note that $pk = (e, n)$). The OT request is sent to T. T computes $\delta = \sigma^d \mod n$ (this can be sped up by the Chinese Remainder Theorem) and sends it to C. C then computes $\gamma = \delta \cdot r^{-1} \mod n$ and the original mapping can be recovered by $z = c \oplus H_2(x, \gamma)$. Algorithm 1 shows the pseudo-code of the entire procedure.

The correctness of the algorithm is ensured by

$$\begin{aligned}
z_i &= c_i \oplus H_2(x_i, \delta_i \cdot r^{-1} \mod n) \\
&= c_i \oplus H_2(x_i, [(r^e \cdot H_1(x_i))^d \mod n] \cdot (r^{-1} \mod n)) \\
&= c_i \oplus H_2(x_i, [(r^{ed} \text{mod } n) \cdot (H_1(x_i)^d \text{mod } n) \cdot (r^{-1} \text{mod } n)]) \\
&= c_i \oplus H_2(x_i, H_1(x_i)^d \mod n) \text{ (since } r^{ed} \mod n = r) \\
&= y_i \oplus H_2(x_i, H_1(x_i)^d \mod n) \oplus H_2(x_i, H_1(x_i)^d \mod n) \\
&= y_i
\end{aligned}$$

Note that $r^{-1} \mod n$ exists if and only if $\gcd(r, n) = 1$ (in which $r^{-1} \mod n$ can be computed by Extended Euclidean Algorithm). Recall that $n = pq$ where p, q are big primes. If $\gcd(r, n) \neq 1$ (namely, p or q), the client is able to factorize n which is believed to be extremely hard. Thus, the probability that a randomly chosen r does not satisfy $\gcd(r, n) = 1$ is negligible. In our experiments, we never observed such a case.[13]

[13]Although we can compute the multiplicative inverse modulo n, it is not possible to derive the private key (d, n) with the public key (e, n) because d is the multiplicative inverse of e modulo $(p-1)(q-1)$ where $n = pq$. Without knowing the factorization of n, i.e., p and q, computing d with e and n is infeasible.

Algorithm 1: Get required mappings from a translator

```
Input: Translator $T$: mappings $x_i \to y_i$, private key $sk = (p, q, n, d)$
Input: Client $C$: indexes of required mappings $M_r$, pub. key $pk$=$(e, n)$
Output: Client $C$: mappings $x_i \to y_i$ where $x_i \in M_r$
// At $T$, encrypting the mappings
for each $x_i \to y_i$ do
    $s_i \leftarrow H_1(x_i)^d \mod n$ ;
    $c_i \leftarrow H_2(x_i, s_i) \oplus y_i$ ;
end
Send all encrypted mappings $x_i \to c_i$ to $C$.
// At $C$, prepare OT requests
for $x_i \in M_r$ do
    $r \leftarrow$ random number in $\mathbb{Z}_n^*$ ;
    $\sigma_i \leftarrow r^e \cdot H_1(x_i) \mod n$ ;
end
Send all $\sigma_i$ to $T$.
// At $T$, answer the OT requests for each OT-Req() do
    $\delta_i \leftarrow \sigma_i^d \mod n$ ;
end
Send back all $\delta_i$ to $C$
// At $C$, recover the mappings
$M \leftarrow \emptyset$ ;
for $x_i \in M_r$ do
    $\gamma_i \leftarrow \delta_i \cdot r^{-1} \mod n$ ;
    $z_i \leftarrow c_i \oplus H_2(x_i, \gamma_i)$ ;
    $M \leftarrow M \bigcup \{x_i \to z_i\}$ ;
end
return $M$
```

B.1 Proof of Theorem 1

Proof: The only message from the client to the translator is $\sigma_i = r^e H_1(x_i) \mod n$ (line 10, Algorithm 1). r is randomly chosen by the client from $\mathbb{Z}_n^* = \{x | x \in [0, n-1] \wedge gcd(x, n) = 1\}$ while e and $H_1(x_i)$ are both known to the translator.

To prove the security of protocol, we show that σ_i appears like a random number to the translator, i.e., Pr$(\sigma_i = a) = \frac{1}{|\mathbb{Z}_n^*|}$ for any fixed $a \in \mathbb{Z}_n^*$.

First, we show that for any fixed $b \in \mathbb{Z}_n^*$, if r is random, then $rb \mod n$ is also random. This is true as $b \in \mathbb{Z}_n^*$ and therefore $gcd(b, n) = 1$. So, the multiplicative inverse $b^{-1} \mod n$ exists. For any fixed $a \in \mathbb{Z}_n^*$, we have:

$$\begin{aligned} \Pr(rb \mod n = a) &= \Pr(r = ab^{-1} \mod n) \\ &= \frac{1}{|\mathbb{Z}_N^*|} \end{aligned}$$

as r is chosen uniformly random from $\mathbb{Z}_N^*$.

A similar logic can be applied to the case that $r^e \mod n$ is random given a random r and a fixed e. Specifically, we choose e, d such that $ed = 1 \mod \phi(n)$ where $\phi(n) = (p = 1)(q-1)$ for $n = pq$. Therefore, there exists some k such that $ed = k\phi(n) + 1$. Moreover, we know that for any element $a \in \mathbb{Z}_n^*$, $(a^e)^d \mod n = a$. This is true as we have $a^\phi(n) \mod n = 1$ and therefore:

$$\begin{aligned} a^{ed} \mod n &= a^{k\phi(n)+1} \mod n \\ &= [(a^\phi(n) \mod n)^k \mod n \times a \mod n] \mod n \\ &= (1^k \mod n \times a) \mod n \\ &= a \mod n = a \end{aligned}$$

Combining the above together, we know that r is random and therefore $r^e \mod n$ is also random. Then, $(r^e \mod n \times H(x_1))) \mod n$ is random. So, σ_i appears random to the translator. As a result, a translator cannot gain any hints about query answer from the client's message. In the mapping request, there are $k - 1$ fake values for each true value in the answer. The probability of each value in the request being a query answer is at most $\frac{1}{k}$. Thus, k-protection is enforced.