Protocols for Secure Remote Database Access with Approximate Matching *

Wenliang Du
CERIAS and
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
Email: duw@cerias.purdue.edu
Tel: (765) 496-6765

Mikhail J. Atallah
CERIAS and
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
Email: mja@cs.purdue.edu
Tel: (765) 494-6017

Florian Kerschbaum
CERIAS and
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
Email: fkerschbaum@cerias.purdue.edu
Tel: (765) 496-6766

Abstract

Suppose that Bob has a database D and that Alice wants to perform a search query q on D (e.g., "is q in D?"). Since Alice is concerned about her privacy, she does not want Bob to know the query q or the response to the query. How could this be done? There are elegant cryptographic techniques for solving this problem under various constraints (such as "Bob should know neither q nor the answer to the query" and "Alice should learn nothing about D other than the answer to the query"), while optimizing various performance criteria (e.g., amount of communication).

We consider the version of this problem where the query is of the type "is q approximately *in D?" for a number of different notions of "approximate", some of which arise in image processing and template matching, while others arise in biological sequence comparisons. New techniques are needed in this framework of approximate searching, because each notion of "approximate equality" introduces its own set of difficulties; using encryption is more problematic in this framework because the items that are approximately equal cease to be so after encryption or cryptographic hashing. Practical protocols for solving such problems make possible new forms of e-commerce between proprietary database owners and customers who seek to query the database, with privacy.*

We first present four secure remote database access models that could be used in an e-commerce framework, each of which has different privacy requirement. We then present our solutions for achieving privacy in each of these four models.

Keywords: Privacy, security, secure multi-party computation, pattern matching, approximate pattern matching

*Portions of this work were supported by Grant EIA-9903545 from the National Science Foundation, and by sponsors of the Center for Education and Research in Information Assurance and Security.

1 Introduction

Consider the following real-life scenario: Alice thinks that she may have some genetic disease, so she wants to investigate it further. She also knows that Bob has a database containing known DNA patterns about various diseases. After Alice gets a sample of her DNA sequence, she sends it to Bob, who will then tell Alice the diagnosis. However, if Alice is concerned about her privacy, the above process is not acceptable because it does not prevent Bob from knowing Alice's private information–both the query and the result.

This kind of situation, which is likely to arise as e-commerce develops, motivates the following general problem formulation:

> *Secure Database Access (SDA) Problem: Alice has a string q, and Bob has a database of strings $T = \{t_1, \ldots, t_N\}$; Alice wants to know whether there exists a string t_i in Bob's database that "matches" q. The "match" could be an exact match or an approximate (closest) match. The problem is how to design a protocol that accomplishes this task without revealing to Bob Alice's secret query q or the response to that query.*

Because of its practical importance and also because not much work has been done for approximate pattern matching in the SDA context, our work particularly focuses on approximate pattern matching.

The exact matching problem has been extensively considered in the literature [20, 5, 17, 16, 21, 23, 22, 12], even though it can theoretically be solved using the general techniques of secure multi-party computation [9]. The motivation for giving these specialized solutions to it is that they are more *efficient* than those that follow from the above-mentioned general techniques. This is also our motivation in considering approximate pattern matching even though it too is a special case of the general secure multi-party computation problem. Unlike exact pattern matching that produces "yes" and "no" answers, approximate pattern matching measures the difference between the two targets, and produces a *score* to indicate how different the two targets are. The metrics used to measure the difference usually are heuristic and are application-dependent. For example, in image template matching [13, 18], $\sum_{i=1}^{n}(a_i - b_i)^2$ and $\sum_{i=1}^{n}|a_i - b_i|$ are often used to measure the difference between two sequences a and b. In DNA sequence matching [14], *edit distance* [1, 6] makes more sense than the above measurements; *edit distance* measures the cost of transforming one given sequence to another given sequence, and its special case, *longest common subsequence* is used to measure how similar two sequences are.

Solving approximate pattern matching problems within the SDA framework is quite a nontrivial task. Consider the $\sum_{i=1}^{n}|a_i - b_i|$ metric as an example. The known PIR (private information retrieval) techniques [20, 5, 17, 16, 21, 23, 22, 12] can be used by Alice to efficiently access each individual b_i without revealing to Bob anything about which b_i (or even which b) Alice accessed (more on this later), but doing this for each individual b_i and then calculating $\sum_{i=1}^{n}|a_i - b_i|$ violates the requirement that Alice should know the total score $\sum_{i=1}^{n}|a_i - b_i|$ *without knowing anything other than that score*, i.e., without learning anything about the individual b_i values. Using a general secure multi-party computation protocol typically does not lead to an efficient solution. The goals of our research, and the results presented in this paper, are finding efficient ways to do such approximate pattern matchings without disclosing private information.

The practical motivations of remote database access do not all point to the model we described in the above SDA formulation. For example, in some situations, Bob's database could be proprietary whereas in some others it could be public (in either case the protocol should reveal nothing to Bob about Alice's query). The "proprietary" nature of a database might make the solution more difficult because Alice should not be able to know more information than the response to her query. There is also another practical framework, within which Alice uses Bob to store a (suitably disguised) version of her private database (a form of outsourcing), and for such a framework the solutions could be quite different. Based on these variants of the problem, we have investigated four SDA models, and defined a class of SDA problems for each

model according to the metrics we use for approximate pattern matching. Of course the difficulties of the problems are not the same for the different metrics, and so far we have solved a subset of those problems. A summary of our results is listed below (the results are stated more precisely in Section 4, and the models are defined in Section 3 – in the meantime see Figure 1 in that section for a summary of each model).

- For the Private Information Matching Model, we have a solution to the approximate pattern matching based on the $\sum_{i=1}^{n}(a_i - b_i)^2$ metric with $O(n * N)$ communication cost, where n is the length of each string and N is the number of strings in the database.
- For the Private Information Matching Model, we also have a solution to the approximate pattern matching based on the $\sum_{i=1}^{n}|a_i - b_i|$ metric using a Monte Carlo technique; the solution gives an estimated result, and it has $O(n * W * N)$ communication cost, where W is a parameter that affects the accuracy of the estimate.
- For the Private Information Matching Model, if we assume that the alphabet is known to the involved parties and its size is finite, we have a solution to approximate pattern matching based on general $\sum_{i=1}^{n} f(a_i, b_i)$ metrics, hence the solutions for the special cases of $\sum_{i=1}^{n}|a_i - b_i|$, $\sum_{i=1}^{n}(a_i - b_i)^2$, and $\sum_{i=1}^{n}\delta(a_i, b_i)$ (where $\delta(x, y)$ is 1 if $x = y$ and 0 otherwise). These solutions have $O(m * n * N)$ communication cost, where m is the number of the symbols in the alphabet. In many cases, m is small. For instance, m is four in DNA databases.
- For the Secure Storage Outsourcing Model, we have a practical solution to approximate pattern matching based on the $\sum_{i=1}^{n}(a_i - b_i)^2$ metric. The solution is practical because its $O(n)$ communication cost does not depend on N.
- For the Secure Storage Outsourcing and Computation Model, we also have a practical solution to approximate pattern matching based on the $\sum_{i=1}^{n}(a_i - b_i)^2$ metric. This solution is practical because of its communication cost is $O(n^2)$.

Motivation

Why do we care about the privacy of a database query? In the example used earlier in this section, if a match is found in the database, Bob immediately knows that Alice has such a disease; even worse, after receiving Alice's DNA sequence, Bob can derive much else about Alice, such as other health problems that Alice might have. If Bob is not trustworthy, Bob could disclose the information about Alice to other parties, and Alice might have difficulty getting employment, insurance, credit, etc. But even if Alice trusts Bob, and Bob has no intention of disclosing Alice's private information, Bob himself might prefer that Alice's query be kept private out of liability concerns: If Bob knows Alice's DNA information, and that information is accidentally disclosed (perhaps by a disgruntled employee of Bob's, or after a system break-in), Bob might face an expensive lawsuit from Alice. From this perspective, a trusted Bob will actually prefer not to know either Alice's query or its response.

With the growth of the Internet, more and more e-commerce transactions like the above will take place. There are already DNA pattern databases, public databases about diseases, patent databases, and in the future we may see many more commercial databases and the related database access services, such as fingerprint databases, signature databases, medical record databases, and many more. Privacy will be a major issue, and assuming the trustworthiness of the service providers, as is done today, is risky; therefore protocols that can support remote access operations while protecting the client's privacy are of growing importance.

One of the fundamental operations behind the queries described in the examples above is pattern matching. Therefore, the basic problem that we face is how to conduct pattern matching operations at the server

side while the server has no knowledge of the client's actual query (or the response to it). In some database access situations, exact pattern matching is used, such as query by name, query by social security number, etc. However, in many other situations, exact pattern matching is unrealistic. For instance, in fingerprint matching, even if two fingerprints come from the same finger, they are unlikely to be exactly the same because there is some information loss in the process of deriving an electronic form (usually a complex data structure of features) from a raw fingerprint image. Similarly in voice, face, and DNA matching; in these and many other situations, exact matching is not expected and some form of approximate pattern matching is more useful.

Background Information on Secure Multi-party Computation

The above problem is a special case of the general secure multi-party computation problem [34]. Generally speaking, a multi-party computation problem deals with computing any probabilistic function on any input, in a distributed network where each participant holds one of the inputs, ensuring independence of the inputs, correctness of the computation, and that no more information is revealed to a participant in the computation than can be computed from that participant's input and output [11]. Other examples of such computations include: elections over the Internet, electronic bidding, joint signatures, and joint decryption. The history of the multi-party computation problem is extensive since it was introduced by Yao [34] and extended by Goldreich, Micali, and Wigderson [24], and by many others: Goldwasser [11] predicts that "the field of multi-party computations is today where public-key cryptography was ten years ago, namely an extremely powerful tool and rich theory whose real-life usage is at this time only beginning but will become in the future an integral part of our computing reality".

Goldreich states in [9] that the general secure multi-party computation problem is solvable in theory. However, he also points out that using the solutions derived by these general results for special cases of multi-party computation, can be impractical; special solutions should be developed for special cases for efficiency reasons.

One of the well-known special cases of multi-party computation is the Private Information Retrieval (PIR) problem: The problem consists of a client and server. The client needs to get the ith bit of a binary sequence from the server without letting the server know the i; the server does not want the client to know the binary sequence either. A solution for this problem is not difficult; however an efficient solution, in particular a solution with small communication cost, is not easy. Studies [20, 5, 17, 16, 21, 23, 22, 12] have shown that one can design a protocol to solve the PIR problem with much better communication complexity than by using the general theoretical solutions. Pattern matching is another such specific computation, and the recent progress in the PIR problem motivated us to speculate that there exist efficient solutions for this particular kind of secure multi-party computation as well.

Secure Multi-party Protocol vs. Anonymous Communication Protocol

Anonymous communication protocols [30, 10] were designed to achieve somewhat related goals, so why not use them? Anonymity techniques help to hide the identity of the information sender, rather than the information being sent. For example, when people browse the web, they can use anonymous communication techniques to keep their identities secret, but the web query usually is not secret because the web server has to know the query in order to send a reply back. In situations where the identity of the information sender needs to be protected, anonymous communication protocols are appropriate. However, there are situations where anonymous communication protocols cannot replace secure multi-party computation protocols. First, certain types of information intrinsically reveal the identity of someone related to the information (e.g., social security number). Secondly, in some situations, it is the information itself that needs to be protected, not the identity of the information sender. For instance, if Alice has an invention, she has to search if such an

invention is new before she files for a patent. When conducting the query, Alice may want to keep the query private (perhaps to avoid part of her idea being stolen by people who have access to her query); she does not care whether her identity is revealed. Thirdly, in certain situations, one has to be a registered member in order to use the database access service; this makes hiding a user's identity difficult because the user has to register and login first, which might already disclose her identity.

Furthermore, most of the known practical anonymous protocols, such as Crowds [30], Onion routing [10] and `anonymizer.com`, use one or several *trusted* third parties. In our secure multi-party computation protocols, we do not use a trusted third party; when a third party is used, we generally assume that the third party is not trusted, and should learn nothing about either Alice's query, or Bob's data, or the response to the query.

Therefore anonymity does not totally solve our problems, and cannot replace secure multi-party computation. Rather, by combining anonymity techniques with secure multi-party computation techniques, one can achieve better overall privacy more efficiently.

2 Related Work

As Goldwasser points out in [11], in the 1980's the focus of research was to show the most general result possible, yielding multi-party protocol solutions for any probabilistic function. Much of the current work is to focus on *efficient* and *non-interactive* solutions to special important problems such as joint-signatures, joint-decryption, and secure and private database access.

Among various multi-party computation problems, the Private Information Retrieval (PIR) problem has been widely studied; it is also the problem most related to what we present in this paper (although here we use none of the elegant techniques for PIR that are found in the literature, for reasons we explained earlier in this paper). The PIR problem consists of devising a protocol involving a user and a database server, each having a secret input. The database's secret input is called the *data string*, an N-bit string $B = b_1 b_2 \ldots b_N$. The user's secret input is an integer i between 1 and n. The protocol should enable the user to learn b_i in a communication-efficient way and at the same time hide i from the database. The trivial solution is having the database send an encryption of the entire string B to the user, with an $O(n * N)$ communication complexity. Much work has been done for reducing this communication complexity [20, 5, 17, 16, 21, 23, 22, 12].

Chor *et al.* point out that a major drawback of all known PIR schemes is the assumption that the user knows the *physical address* of the sought item [8], whereas in the current database query scenario the user typically holds a keyword and the database internally converts this keyword into a physical address. To solve this problem, Chor *et al.* propose a scheme to privately access data by keywords [8]. The difference between the problem studied in Chor's paper and the problems in our paper is that we extend the problem to cover approximate pattern matching.

Song *et al.* propose a scheme to conduct searches on encrypted data [33]. In that framework, Alice has a database, and she has to store the database in a server controlled by Bob; how could Alice query her database without letting Bob know the contents of the database or the query? Here we primarily focus on extending the problem to also cover approximate pattern matching.

3 Framework

3.1 Models

Remote database access has many variants. In some e-commerce models, Bob's database is private while in some other models, it is public. In the latter case, there is no requirement to keep the database secret from Alice; however, the privacy of Alice's query still needs to be preserved. In other e-commerce models, Bob

hosts Alice's (encrypted/disguised) database while supporting queries from Alice and other customers, in which case Bob should know neither the database nor the queries.

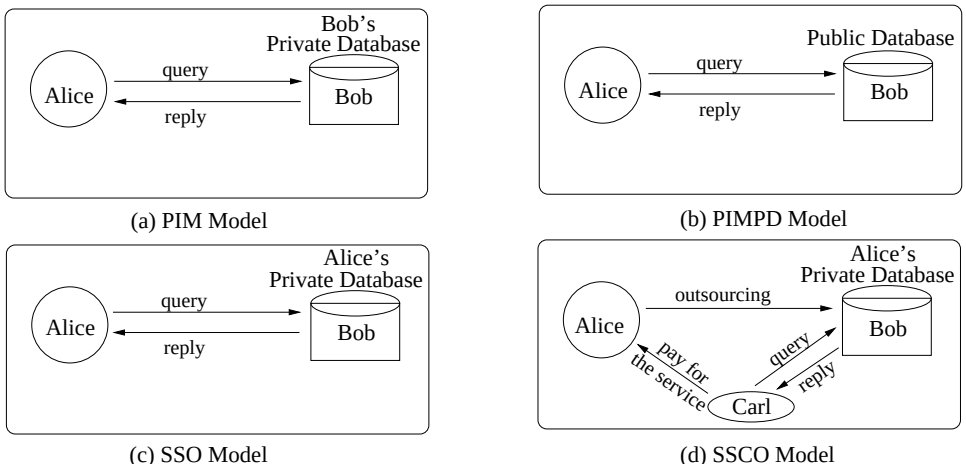

Figure 1: Models

From the various ways that remote database access is conducted, we distinguish four different e-commerce models, all of which require customers' privacy:

- PIM: Private Information Matching Model (Figure 1.a)
- PIMPD: Private Information Matching from Public Database Model (Figure 1.b).
- SSO: Secure Storage Outsourcing Model (Figure 1.c).
- SSCO: Secure Storage and Computing Outsourcing Model (Figure 1.d).

For the sake of convenience, we will use $Match()$ to represent the pattern matching function, which includes both exact pattern matching and approximate pattern matching.

Private Information Matching Problem (PIM)

Alice has a string x, and Bob has a database of strings $T = \{t_1, \ldots, t_N\}$; Alice wants to know the result of $Match(x, T)$. Because of the privacy concern, Alice does not want Bob to know the query x or the response to the query; Bob does not want Alice to know any string in the database except for what can be derived from the reply. Furthermore, Bob wants to make money from providing such a service, therefore Alice should not be able to conduct the querying by herself; in other words, every time Alice wants to perform such a query, she has to contact Bob, otherwise she cannot get the correct answer.

Private Information Matching from Public Database Problem (PIMPD)

Bob has a database of strings $T = \{t_1, \ldots, t_N\}$, whose contents are public knowledge. Alice has a query x, and she wants to know the result of $Match(x, T)$ without disclosing to Bob either her query x or the response to it.

This problem is different from the PIM problem: In the PIM problem, Bob does not allow Alice to know any information about the database except for what can be derived from the reply. In the PIMPD problem, since the database contains only public knowledge, there is no need to prevent Bob from letting Alice know

more about the contents of the database than the strict answer to her query (although Bob's doing so may result in unnecessary communication).

Secure Storage Outsourcing Problem (SSO)

Alice has a database of strings $T = \{t_1, \ldots, t_N\}$, but she does not have enough storage for the large database, so she outsources her database (suitably disguised–more on this later) to Bob, who provides enough storage for Alice. Furthermore, from time to time, Alice needs to query her database and retrieves the information that matches her query, i.e., Alice wants to know $Match(x, T)$ for her query x. As usual, Alice wants to keep the contents of both the database and the query secret from Bob.

Secure Storage and Computing Outsourcing Problem (SSCO)

The SSCO problem is an extension of the SSO problem. Whereas only Alice queries her database in the SSO problem, in the SSCO model the database will also be queried by other clients of Alice. More specifically, in the SSCO model, Alice outsources her database to Bob, and she wants the database to be available to anyone who is willing to pay her for the database access service. When a client accesses the database, neither Alice nor Bob should know the contents of the query. Moreover, Alice wants to charge the clients for each query they have submitted, so the client should not be able to get the correct query result if Alice is not aware of the query's existence.

Since Bob can pretend to be a client, the solutions of the SSCO problem should be secure even if Bob can collude against Alice with any client. However, the SSO problem does not have such a concern because the only client is Alice herself.

3.2 Notation

For each model, there is a family of problems. We will use the following notations to represents each specific problem:

- M/Exact: Exact Pattern Matching problem in model M.
- M/Approx: Approximate Pattern Matching problem in model M.
  - M/Approx/f: use $\sum_{k=1}^{n} f(a_k, b_k)$ metric to measure the distance between two strings, where f is a general function.
  - M/Approx/δ: use $\sum_{k=1}^{n} \delta(a_k, b_k)$ metric to measure the distance between two strings, where δ is the Kronecker symbol: $\delta(x, y)$ = 0 if and only if $x = y$ and 1 otherwise.
  - M/Approx/Abs: use $\sum_{k=1}^{n} |a_k - b_k|$ metric to measure the distance between two strings.
  - M/Approx/Squ: use $\sum_{k=1}^{n} (a_k - b_k)^2$ metric to measure the distance between two strings.
  - M/Approx/Edit:
    - M/Approx/Edit/String: use the string editing criterion [6] to measure the distance between two strings.
    - M/Approx/Edit/Tree: use the tree editing criterion to measure the distance between two trees.

The M/Exact problem has been studied extensively in certain model, such as PIM and SSO, but the M/Approx problem has not. Our results deal mostly with the M/Approx problem.

4 Protocols Using Untrusted Third Party

This section presents protocols that use an untrusted third party. The next section (Section 5) shows how to get rid of Ursula and get strictly 2-party solutions, but at the considerable cost of repeatedly using Yao's millionaire protocol [34], and also the (somewhat) expensive oblivious transfer technique [29, 26].

4.1 Preliminary

Protocol for scalar (and other) products

Recall that the scalar product of two vectors $\vec{x} = (x_1, ..., x_n)$ and $\vec{y} = (y_1, ..., y_n)$ is:
$\vec{x} \cdot \vec{y} = \sum_{k=1}^{n} x_k * y_k$.

We describe a protocol for Alice and Bob to compute the scalar product of Alice's vector $\vec{x}$ and Bob's vector $\vec{y}$ using an untrusted third party Ursula. Neither Alice nor Bob should learn anything about the other party's input (other than what can be derived from knowing $\vec{x} \cdot \vec{y}$), and Ursula should learn nothing about either $\vec{x}$ or $\vec{y}$. This protocol will later serve as a building block for other protocols. Essentially the same protocol also solves the asymmetric version of the problem, in which only Alice is to know $\vec{x} \cdot \vec{y}$.

1. Alice and Bob jointly generate two random numbers r and r'.
2. Alice and Bob jointly generate two random vectors $\vec{R}$, $\vec{R}'$ (of size n).
3. Alice sends $\vec{w}_1 = \vec{x} + \vec{R}$ and $s_1 = \vec{x} \cdot \vec{R}' + r$ to Ursula.
4. Bob sends $\vec{w}_2 = \vec{y} + \vec{R}'$ and $s_2 = \vec{R} \cdot (\vec{y} + \vec{R}') + r'$ to Ursula.
5. Ursula computes $v = \vec{w}_1 \cdot \vec{w}_2 - s_1 - s_2$ and gets $v = \vec{x} \cdot \vec{y} - (r + r')$; she then send the result to Alice and Bob. *Note.* In the asymmetric version of the problem (in which only Alice is to know $\vec{x} \cdot \vec{y}$) Ursula does not send anything to Bob in this step.
6. Alice and (in the symmetric version of the problem) Bob then get $\vec{x} \cdot \vec{y} = v + (r + r')$.

Note that the communication complexity of the above is linear in the size of the inputs, i.e., it is $O(n)$.

Although only the scalar product case is needed later in this paper, it should be clear that other operations than scalar product can be carried out using suitably modified versions of the above protocol. These include matrix product, in which case Alice and Bob have matrices and R, R', r, r' are random matrices. They also include the convolution product of two vectors, in which case R, R', r, r' are random vectors. This could potentially be useful in other contexts.

A less practical protocol that does not require using Ursula will be given in Section 5.

4.2 PIM/Approx

Except for the research on the general secure multi-party computation problem, this specific problem has not been studied in the literature. Unless otherwise specified, we assume the alphabet used in the following solution to be predefined and its size to be finite. This assumption is quite reasonable in many situations; for instance, DNA sequences use a fixed alphabet of four symbols. Under this assumption, we can solve the PIM/Approx/f problem. However, because the way to calculate *edit distance* cannot be represented in the form $\sum_{k=1}^{n} f(a_k, b_k)$, the PIM/Approx/Edit problem is not a special case of the PIM/Approx/f problem.

In some other situations, the above finite alphabet assumption does not apply. For instance, fingerprint, image and voice patterns use real numbers instead of characters from a known finite alphabet. The above-mentioned solution for the PIM/Approx/f problem cannot be used anymore, however by exploiting the

mathematical property of $\sum_{i=1}^{n}(a_i - b_i)^2$, we have come up with a solution for the PIM/Approx/Squ problem for infinite alphabet after introducing an *untrusted* third party who does not know the inputs from either of the two parties and learns nothing about them (or about the query, or the answer to it). We also have a solution to the PIM/Approx/Abs problem using a Monte Carlo technique. All of these are given below.

4.2.1 PIM/Approx/Squ Protocol

Suppose that Bob has a database $T = \{t_1, ..., t_N\}$, and assume the length of each string t_i is n; Alice wants to know the $t_i \in T$ that most closely matches a query $x = x_1...x_n$ based on the PIM/Approx/Squ metric. The requirement is that Bob should not know x or the result, and Alice should not be able to learn more information than the reply from Bob.

We propose a protocol to compute the matching score using an untrusted third party, Ursula. Our assumption here is that Ursula will not conspire with either Alice or Bob. However, the third party is not fully trusted: Ursula should not be able to deduce either x or T, or the final matching score s. This protocol works for both finite and infinite alphabet.

Let $\vec{x} = (-2x_1, ..., -2x_n, 1)$; for each $t_i = y_{i,1}...y_{i,n}$, let $\vec{z_i} = (y_{i,1}, ..., y_{i,n}, \sum_{k=1}^{n} y_{i,k}^2)$. Observe that:

$$\sum_{k=1}^{n}(x_k - y_{i,k})^2 = \vec{x} \cdot \vec{z_i} + \sum_{k=1}^{n} x_k^2.$$

Since $\sum_{k=1}^{n} x_k^2$ is a constant, we can use $\vec{x} \cdot \vec{z_i}$ instead of $\sum_{k=1}^{n}(x_k - y_{i,k})^2$ to find the closest match. After we get the closest match, Alice can calculate the actual score by adding $\sum_{k=1}^{n} x_k^2$.

Protocol

1. Alice and Bob jointly generate two random numbers r and r'.
2. For each $t_i \in T$, repeat the next five sub-steps, in which $t_i = y_{i,1}...y_{i,n}$, $\vec{x} = (-2x_1, ..., -2x_n, 1)$.
   (a) Bob constructs $\vec{z_i} = (y_{i,1}, ..., y_{i,n}, \sum_{k=1}^{n} y_{i,k}^2)$.
   (b) Alice and Bob jointly generate two random vectors $\vec{R}$, $\vec{R'}$ (of size $n+1$).
   (c) Alice sends $\vec{w_1} = \vec{x} + \vec{R}$ and $s_1 = \vec{x} \cdot \vec{R'} + r$ to Ursula.
   (d) Bob sends $\vec{w_2} = \vec{z_i} + \vec{R'}$ and $s_2 = \vec{R} \cdot (\vec{z_i} + \vec{R'}) + r'$ to Ursula.
   (e) Ursula computes $v_i = \vec{w_1} \cdot \vec{w_2} - s_1 - s_2$ and gets the resulting $v_i = \vec{x} \cdot \vec{z_i} - (r + r')$.
3. Ursula computes $score' = \min_{i=1}^{N} v_i$, and sends the resulting $score'$ to Alice.
4. Alice computes $score = score' + \sum_{k=1}^{n} x_k^2 + (r + r')$, which is the closest match between x and any $t_i \in T$.

The random vectors $\vec{R}$ and $\vec{R'}$ are used to disguise Alice's and Bob's data; the random numbers r and r' are used to disguise the query results and the intermediate results. The communication cost is $O(n * N)$.

4.2.2 PIM/Approx/Abs Protocol

First, we will present a Monte Carlo technique for Alice and Bob to calculate $|x_k - y_k|$ (x_k is Alice's secret input and y_k is Bob's), and then use it as a building block to compute $\sum_{k=1}^{n} |x_k - y_k|$. The protocol involves an untrusted third party, Ursula, who learns nothing. The protocol works for both finite and infinite

alphabets. Assume that $0 < x_k \leq U$ and $0 < y_k \leq U$ for some number U. The protocol for $|x_k - y_k|$ is as follows (where W is a parameter that affects the accuracy of the estimate, and $counter = 0$ initially):

1. Alice generates a random number R_k, and then generates a sequence of $W - R_k$ random i.i.d. numbers, each uniformly over $(0..U]$.
2. Alice randomly replaces half of these $W - R_k$ numbers with their negative values.
3. Alice "splices" R_k zeroes into random positions of the above sequence of $W - R_k$ numbers, resulting in a new sequence S of W numbers.
4. Alice then sends S to Bob.
5. For each number s from S, if $s = 0$, Alice sends 1 to Ursula; if $s > 0$ then Alice sends 1 to Ursula if $|s| \geq x_k$ and sends 0 otherwise; if $s < 0$ then Alice sends 0 to Ursula if $|s| \geq x_k$ and sends 1 otherwise.
6. For each number s from S, if $s = 0$, Bob sends 0 to Ursula; if $s > 0$ then Bob sends 1 to Ursula if $|s| \geq y_k$ and sends 0 otherwise; if $s < 0$ then Bob sends 0 to Ursula if $|s| \geq y_k$ and sends 1 otherwise.
7. Ursula increases $counter$ by 1 if the values she receives from Alice and Bob are different.
8. Ursula computes $score = counter * \frac{U}{W}$, which is an unbiased estimate of $|x_k - y_k| + R_k * \frac{U}{W}$.

Because of R_k, Ursula does not know the actual distance between x_k and y_k, and because of the negative numbers among those W random numbers, Ursula cannot figure out whether $x_k > y_k$ or $x_k < y_k$.

Now, let us see how to use the above protocol to compute $\sum_{k=1}^{n} |x_k - y_{i,k}|$, where $x = x_1...x_n$ and $t_i = y_{i,1}...y_{i,n}$:

1. Alice generates a random number R.
2. For each $t_i \in T$, suppose $t_i = y_{i,1}...y_{i,n}$ and repeat the next three sub-steps:
   (a) $counter = 0$.
   (b) For each $k = 1, ..., n$, Alice, Bob and Ursula use the above protocol to compute $|x_k - y_{i,k}|$. The random numbers $R_{i,1}, ..., R_{i,n}$ used in the above protocol are generated by Alice, such that $\sum_{k=1}^{n} R_{i,k} = R$.
   (c) Ursula computes $score_i$ = $counter * \frac{U}{W}$, which is an unbiased estimate of $\sum_{k=1}^{n} |x_k - y_{i,k}| + \sum_{k=1}^{n} R_{i,k} * \frac{U}{W}$ = $\sum_{k=1}^{n} |x_k - y_{i,k}| + R * \frac{U}{W}$.
3. Ursula computes $score' = \min_{i=1}^{N} score_i$, and sends $score'$ to Alice.
4. Alice computes $score = score' - R * \frac{U}{W}$ and gets the closest match between x and any $t_i \in T$.

The communication complexity is $O(n * W * N)$.

4.2.3 PIM/Approx/f protocol

If the alphabet is predefined and its size is finite, we can solve a general problem–computing $f(x_k, y_k)$. However, we cannot directly use this protocol n times to compute $\sum_{k=1}^{n} f(x_k, y_k)$ because that would reveal each individual $f(x_k, y_k)$ result. We will present the protocol for computing $f(x_k, y_k)$ here, and then in the following sub-section, we will discuss how to use it as a building block to compute $\sum_{k=1}^{n} f(x_k, y_k)$ without revealing any individual $f(x_k, y_k)$.

Suppose Alice has an input x_k; Bob has an input y_k; Alice wants to know the result of $f(x_k, y_k)$ without revealing x_k and the result to Bob, and Bob does not want to reveal his y_k to Alice. After presenting a solution to this problem, we later use it as a building block to construct solutions to other problems.

f-function Protocol

We assume the encryption methods used below are commutative.

1. Bob computes $f(\alpha_i, y_k)$ for each $\alpha_i \in X$, where X is the finite (known) alphabet. Let m be the size of X.
2. Bob chooses a secret key k, computes $E_k(f(\alpha_i, y_k))$ for each $\alpha_i \in X$, and sends to Alice the m results.
3. Alice chooses one from $E_k(f(\alpha_i, y_k))$, $i = 1 \ldots m$, such that $\alpha_i = x_k$. This can be done because Bob sent the m encrypted results in order.
4. Alice chooses a secret key k', computes $E_{k'}(E_k(f(x_k, y_k)))$, and sends it back to Bob.
5. Because of the commutative properties of $E_{k'}$ and E_k, $E_{k'}(E_k(f(x_k, y_k)))$ is equivalent to $E_k(E_{k'}(f(x_k, y_k)))$, which could be decrypted to $E_{k'}(f(x_k, y_k))$ by Bob. Bob sends the result $E_{k'}(f(x_k, y_k))$ to Alice.
6. Alice gets $f(x_k, y_k)$ by decrypting $E_{k'}(f(x_k, y_k))$.

The technique used above is similar to the standard oblivious transfer protocol; it protects the privacy of the inputs from both parties without introducing a third-party. The communication cost is $O(m)$, where m is the size of the alphabet.

PIM/Approx/f Protocol

Now, let us see how to securely compute $\min_{i=1}^{N}(\sum_{k=1}^{n} f(x_k, y_{i,k}))$. As we discussed above, we cannot run the above f-function protocol n times to get $\sum_{k=1}^{n} f(x_k, y_{i,k})$. In the following protocol, we will use a disguise technique to hide each individual result of $f(x_k, y_{i,k})$.

For each $t_i = y_{i,1}...y_{i,n}$, and for each $k = 1, ..., n$, let $f_{i,k}(x_k, y_{i,k}) = f(x_k, y_{i,k}) + R_{i,k}$, where $R_{i,k}$ is a random number, the following protocol shows how A and B calculate $\min_{i=1}^{N} \sum_{k=1}^{n} f(x_k, y_{i,k})$:

1. Bob generates a random number R then sends R to Alice.
2. For each $t_i = y_{i,1}, ..., y_{i,n}$, repeat the next five sub-steps:
   (a) Bob constructs $f_{i,k}(x_k, y_{i,k}) = f(x_k, y_{i,k}) + R_{i,k}$ for $k = 1, ..., n$, where $R_{i,1}, ..., R_{i,n}$ are n random numbers.
   (b) Alice and Bob use the f-function protocol to compute $f_{i,k}(x_k, y_{i,k})$, for each $k = 1, ..., n$.

(c) Alice sends $\sum_{k=1}^{n} f_{i,k}(x_k, y_{i,k})$ to Ursula.

(d) Bob sends $\sum_{k=1}^{n} R_{i,k} - R$ to Ursula.

(e) Ursula computes $score_i = \sum_{k=1}^{n} f_{i,k}(x_k, y_{i,k}) - (\sum_{k=1}^{n} R_{i,k} - R) = \sum_{k=1}^{n} f(x_k, y_{i,k}) + R$.

3. Ursula computes $score' = \min_{i=1}^{N} score_i$, and sends $score'$ to Alice.

4. Alice compute $score = score' - R$, thus getting the actual distance between x and the closest t_i in the database T.

Although Alice knows each individual $f_{i,k}(x_k, y_{i,k})$, she does not know the actual value of $f(x_k, y_{i,k})$ because of $R_{i,k}$. Similarly, because of R, Ursula does not know the actual score of the closest match. The communication cost of the protocol is $O(m * n * N)$, where m is the size of the alphabet, n is the length of each pattern, and N is the size of the database. In many cases, m is quite small. For instance, m is four in DNA databases.

Because $|x_k - y_k|$, $(x_k - y_k)^2$ and $\delta(x_k, y_k)$ functions are special cases of $f(x_k, y_k)$, PIM/Approx/(Abs, Squ, δ) problems can all be solved using the above protocol.

4.3 PIMPD/Approx

The only difference between the PIM model and the PIMPD model is that, in the latter, Bob does not need to keep the database secret from Alice. Therefore, all solutions in the PIM model can be applied to the PIMPD model as well. Whether the "public" feature of the database can result in more efficient solutions is an interesting question. Although we do not yet have an answer to it, we observed the following:

Observation 1. *There is no secure two-party non-interactive solution for the PIMPD/Approx problem.*

Proof. A two-party non-interactive protocol means Bob, by himself, is able to find the item in the database that has minimal distance from the query.

Assume there is a two-party non-interactive protocol A which solves any of the PIMPD/Approx problems, in another words, given an encrypted/disguised form (q') of a query q, and the database T that Bob knows, Bob can find the item in the database that has minimal distance from q as follows. We use $A(T, q')$ to represent the algorithm on input T and q'.

Since Bob can use any database he wants, he can use a database like this: T' = {"axxxxxx", "bxxxxxx", ..., "zxxxxxx"}, supposing that the alphabet is a set from 'a' to 'z'. After applying $A(T', q')$, Bob will get one that has the minimal distance from q. For instance, if "mxxxxxx" is the result, Bob knows that 'm' is the first character in q. Since A is a non-interactive protocol, Bob can reuse it on another database constructed for the purpose of exposing the second character in q; he can keep doing this and figure out the rest of the characters in q.

Therefore, if such a protocol existed, the query q would not be kept secret from Bob. □

The above observation does not rule out the existence of an efficient interactive protocol or a multi-party protocol.

4.4 SSO/Approx

In this model, Bob is a service provider who provides storage and database query services to Alice. According to Alice's privacy requirement, Bob should know nothing about the database that he stores for Alice, nor should he know the query. So Bob has to conduct a disguised database query based on the encrypted or disguised data of Alice.

The requirement that Bob should not know the query result, as in the PIM and PIMPD problem, is no longer needed in the SSO problem. The reason is that Bob does not know the contents of the database, he does not even know what the database is for, so that knowing whether Alice's query is in the database does not disclose any secret information to Bob.

Intuitively, it can look like that the SSO/Approx problem might be more difficult than the PIM/Approx problem because in the latter Bob at least knows the contents of the database whereas in the former he knows nothing about the database. But knowing the contents of the database has a disadvantage, in that Bob cannot know an intermediate result because he knows one of the inputs (the database); if he also knew an intermediate result, he might be able to figure out the other input (the query) of the computation. However, in the SSO/Approx problem, Bob knows nothing about the database, so it is safe for him to know intermediate results without exposing the secret query.

Whether Bob can know intermediate results is a critical issue for reducing the communication complexity. If he knew intermediate results to some extent, he could conduct the comparison operation to find the minimal or maximal score; otherwise, he has to turn to Alice in order to find the minimal or maximal score, which results in high communication cost in the PIM problem.

The SSO/Approx problem is similar to the secure outsourcing of scientific computations problems studied by Atallah *et al.* [3]. The difference is that in secure outsourcing problems, the inputs are provided by Alice every time a computation is conducted at Bob's side; therefore, Alice can encrypt/disguise the inputs differently in different rounds of the computation. However, in the SSO problem, one of the inputs (the database) is encrypted/disguised only once, and this same input is used in all rounds of computations; this makes the problem more difficult.

So far, we have a solution only for SSO/Approx/Squ problem. The solution works for both infinite and finite alphabets.

4.4.1 SSO/Approx/Squ Protocol

Suppose that Alice wants to outsource her database $T = \{t_1, ..., t_N\}$ to Bob, and wants to know if query string $x = x_1...x_n$ matches any pattern t_i in the database T.

The straightforward solution would be to let Bob send the whole database back to Alice, and let Alice conduct the query by herself. Although this solution satisfies the privacy requirement, much better communication complexity can be achieved. Another intuitive question would be whether Bob can conduct the matching independently after Alice sends him the relevant information about the query. If the answer is true, Bob should be able to find the item t_i that has the closest match to the query x. In another words, if $t_i = y_1...y_n$ and $score_i = \sum_{k=1}^{n}(x_k - y_k)^2$, then Bob should be able to find the minimum value of $score_i$. However, because of the privacy requirement, Bob is not allowed to know the actual query x, nor is he allowed to know the content of the database, so how does he compute the distance $score_i$ between x and each of the element t_i in the database?

The idea behind our solution is based on the fact that $\vec{x} \cdot \vec{z}^T = (\vec{x}Q^{-1}) \cdot (Q\vec{z}^T)$, where Q is an invertible matrix. Alice can store $Q\vec{z}^T$ instead of $\vec{z}^T$ at Bob's site, and keeps Q secret from Bob. She will send $\vec{x}Q^{-1}$ to Bob each time she wants to send a query x; therefore Bob can compute $\vec{x} \cdot \vec{z}^T$ without even knowing $\vec{x}$ and $\vec{z}$. If we can use $\vec{x} \cdot \vec{z}^T$ to represent the $\sum_{k=1}^{n}(x_k - y_k)^2$, we can make it possible for Bob to conduct the approximate pattern matching.

For each $t_i = y_{i,1}...y_{i,n}$ in the database T, let $\vec{t_i} = (\sum_{k=1}^{n} y_{i,k}^2 + R - R_i, y_{i,1}, ..., y_{i,n}, 1, R_i)$, and let $\vec{x} = (1, -2x_1, ..., -2x_n, R_A, 1)$, where R, R_A and R_i are random numbers. We will have $\vec{x} \cdot \vec{t_i}^T = \sum_{k=1}^{n} y_{i,k}^2 - 2\sum_{k=1}^{n} x_k y_{i,k} + R + R_A$, and therefore $score_i = \sum_{k=1}^{n}(x_k - y_{i,k})^2 = \vec{x} \cdot \vec{t_i}^T + (\sum_{k=1}^{n} x_k^2 - R - R_A)$. Since $(\sum_{k=1}^{n} x_k^2 - R - R_A)$ is a constant, it does not affect the final result if we only want to find the t_i that produces the minimum $score_i$. Therefore, Bob can use $\vec{x} \cdot \vec{t_i}^T$ to compute the closest match.

Before outsourcing the database to Bob, Alice randomly chooses a secret $(n+3) \times (n+3)$ invertible matrix Q, and computes $\vec{z}_i = Q\vec{t}_i^T$, then sends $T' = \{\vec{z}_1, ..., \vec{z}_N\}$ to Bob.

Protocol

1. For any query string $x = x_1...x_n$, Alice generates a random number R_A, and constructs a vector $\vec{x} = (1, -2x_1, ..., -2x_n, R_A, 1)$, then sends $\vec{x}Q^{-1}$ to Bob.
2. Bob computes $score'_i = \vec{x} \cdot \vec{z}_i^T$, for $i = 1, ..., N$.
3. Bob computes $\min_{i=1}^{N} score'_i$, and gets the corresponding i.
4. Bob returns $\vec{z}_i$ to Alice.
5. Alice computes $Q^{-1}\vec{z}_i$ and gets t_i, which is the closest match of her query.

Because Alice and Bob are involved in only one round of communication, the communication cost is $O(n)$.

Notice that we have introduced random numbers R, R_A, R_i for $i = 1, ..., N$. The purpose of R is to prevent Bob from knowing the actual distance between x and the items in the database; the purpose of R_A is to prevent Bob from knowing the relationship between two different queries; the purpose of R_i is to prevent Bob from knowing the relationship among items in the database. Without R_i, two similar items in the database T would still be similar to each other in the disguised database T'; adding a different random number to each different item will make this similarity disappear.

4.5 SSCO/Approx

This model poses more challenges than the SSO model becase Bob could now collude against Alice with a client, or he can even become a client. Therefore, one of the threats would be for Bob to compromise the privacy of the database by conducting a number of queries and deriving the way the database is encrypted or disguised. A secure protocol should resist this type of active attack. We have a solution for the SSCO/Approx/Squ problem that works for both infinite and finite alphabets.

4.5.1 SSCO/Approx/Squ protocol

One of the differences between the SSCO/Approx problem and the SSO/Approx problem is who sends the query. In the SSO/Approx/Squ protocol, Alice transforms the query x to a vector $\vec{x}Q^{-1}$, and sends the vector to Bob; in the SSCO/Approx/Squ protocol, the client Carl will send the query. Because Carl does not know Q, he cannot construct $\vec{x}Q^{-1}$ by himself. If Carl could get the result of $\vec{x}Q^{-1}$ securely, namely without disclosing $\vec{x}$ to Alice and without knowing Q of course, we would have a solution. Because $Q^{-1} = (\vec{q}_1^T, ..., \vec{q}_m^T)$, computing $\vec{x}Q^{-1}$ securely is basically a task of computing $\vec{x} \cdot \vec{q}_k^T$ for $k = 1 \ldots m$, which can be solved using the same technique as that used in solving PIM/Approx/Squ problem.

Therefore, by modifying step 2 of the SSO/Approx/Squ protocol slightly, and also by using a form of "$R_\alpha * (score + R_A)$", instead of the form of "$score + R_A$" as is used in SSO/Approx/Squ protocol, we obtain a SSCO/Approx/Squ protocol as the following:

Let $T = \{t_1, ..., t_N\}$ be the database Alice wants to outsource to Bob, and assume the length of each string t_i is n. Alice generates N random numbers $R_1, ..., R_N$. For each $t_i = y_{i,1}, ..., y_{i,n}$, let $\vec{t}_i = (\sum_{k=1}^{n} y_{i,k}^2 + R - R_i, y_{i,1}, ..., y_{i,n}, 1, 1, R_i)$; let $\vec{z}_i = Q\vec{t}_i^T$, where Q is a randomly generated $(n+4) \times (n+4)$ matrix.

In what follows, we assume that Alice outsourced the database $T' = \{\vec{z}_1, ..., \vec{z}_N\}$ to Bob.

Protocol

1. Whenever a client Carl wants to conduct a search on query $x = x_1...x_n$, he generates a random number R_C.
2. Alice generates random numbers R_A and R_α.
3. Carl and Alice jointly compute $\vec{q} = R_\alpha \vec{x} Q^{-1}$, where $\vec{x} = (1, -2x_1, ..., -2x_n, R_C, R_A, 1)$. The computation does not reveal Alice's secret Q, R_A or R_α to Carl, nor does it reveal Carl's private query x or R_C to Alice.
4. Carl then sends the vector $\vec{q}$ to Bob.
5. Bob computes $score_i = \vec{q} \cdot \vec{z}_i^T = R_\alpha(\sum_{k=1}^n y_{i,k}^2 - 2\sum_{k=1}^n x_k y_{i,k} + R_C + R_A)$
6. Bob returns to Alice $score' = \min_{i=1}^N score_i$.
7. Alice computes $score'' = \frac{score'}{R_\alpha} - R_A = \sum_{k=1}^n y_{i,k}^2 - 2\sum_{k=1}^n x_k y_{i,k} + R_C$ and sends it to Carl.
8. Carl computes $score = score'' + \sum_{k=1}^n x_k^2 - R_C$, which is the answer he seeks.

Because of R_C, Alice cannot figure out the actual score for this query, and because of R_A and R_α, Carl cannot figure out the actual score between his query and other items in the database (except for the matched one), even if Carl could collude with Bob. The communication cost of the protocol is $O(n^2)$, most of which is contributed by the computation of $R_\alpha \vec{x} Q^{-1}$ in step 3.

5 Doing Without the Third Party

In this section we present techniques for avoiding the use of the third party Ursula. We illustrate these by giving a 2-party (without Ursula) solution to the PIM/Approx/Squ problem. The reader can verify that they can also get rid of Ursula for the other protocols as well. The solutions that use these techniques are, however, less practical than those of the previous section (that used Ursula) mostly because of the repeated use of Yao's millionaire protocol [34] (recall that, in this protocol, Alice and Bob have a number each and seek to determine which one has the larger number without revealing anything else about their respective numbers). If a practical 2-party protocol (i.e., without an Ursula) to Yao's millionaire problem is ever found, then the techniques we describe in this section could become more practical. Another cost incurred here is that of an oblivious transfer protocol, in which Bob has N items and Alice is to receive one of them without Bob knowing which one she received [29, 26].

Recall that, in the PIM/Approx/Squ problem, Bob has a database $T = \{\vec{t}_1, ..., \vec{t}_N\}$, where the length of each string $\vec{t}_i$ is n, and Alice wants to know the $\vec{t}_i \in T$ that most closely matches a query $\vec{x} = x_1...x_n$ based on the PIM/Approx/Squ metric.

The new protocol consists of two steps:

1. Using the 2-party modified scalar product protocol (given below), Alice obtains N of the form scalars $a_i = \vec{t}_i \cdot \vec{x} + b_i$, where b_i is a random scalar known to Bob only and whose purpose is to hide $\vec{t}_i \cdot \vec{x}$ from Alice (who is supposed to learn only the smallest of the $\vec{t}_i \cdot \vec{x}$ products). After this, Alice has a vector $\vec{a} = (a_1, \ldots, a_N)$ and Bob has a vector $\vec{b} = (b_1, \ldots, b_N)$. Note that the problem of computing the smallest $\vec{t}_i \cdot \vec{x}$ is the same as that of computing the smallest element in the vector $\vec{s} = \vec{a} - \vec{b}$. Alice has to know this smallest element without Bob learning the answer (or the position j in $\vec{s}$ at which the minimum occurs).
2. Using the "find minimum" protocol given below, Alice (but not Bob) gets the minimum element of $\vec{s}$.

5.1 Protocol for Modified Scalar Product

Alice has a vector $\vec{x}$, Bob has a vector $\vec{y}$ and a scalar b, and the goal is for Alice (but not Bob) to know $\vec{x} \cdot \vec{y} + b$, without Alice learning anything new (other than what can be inferred from $\vec{x} \cdot \vec{y} + b$).

Alice could send p vectors to Bob, with one of them equal to $\vec{x}$ and the others being "fake" vectors (structured like $\vec{x}$, e.g., random if $\vec{x}$ is random, etc). Then Bob computes the modified scalar product $\vec{w} \cdot \vec{y} + b$ for each of these p vectors $\vec{w}$, and Alice then uses a oblivious transfer to retrieve the answer that corresponds to the true $\vec{x}$. However if $\vec{x}$ has certain properties known to Bob, he might be able to differentiate $\vec{x}$ out of the other $p - 1$ seemingly similar "fake" vectors. A "one out of p" degree of uncertainty for Bob is also unacceptably small.

The above drawbacks can be fixed by dividing vector $\vec{x}$ into m random vectors $\vec{u}_1, \ldots, \vec{u}_m$, with $\vec{x} = \sum_{j=1}^{m} \vec{u}_j$. The same method as described above can then be used to let Alice know $\vec{u}_j \cdot \vec{y} + r_j$, where r_j is a random number, and hence $\sum_{j=1}^{m} r_j = b$ (see Figure 2). Certainly, there is a 1 out of p possibility that Bob can guess the correct $\vec{u}_j$, but since $\vec{x}$ is the sum of m such random vectors, the chance that Bob guesses the correct $\vec{x}$ is 1 out of p^m, which is very small if we choose p^m large enough.

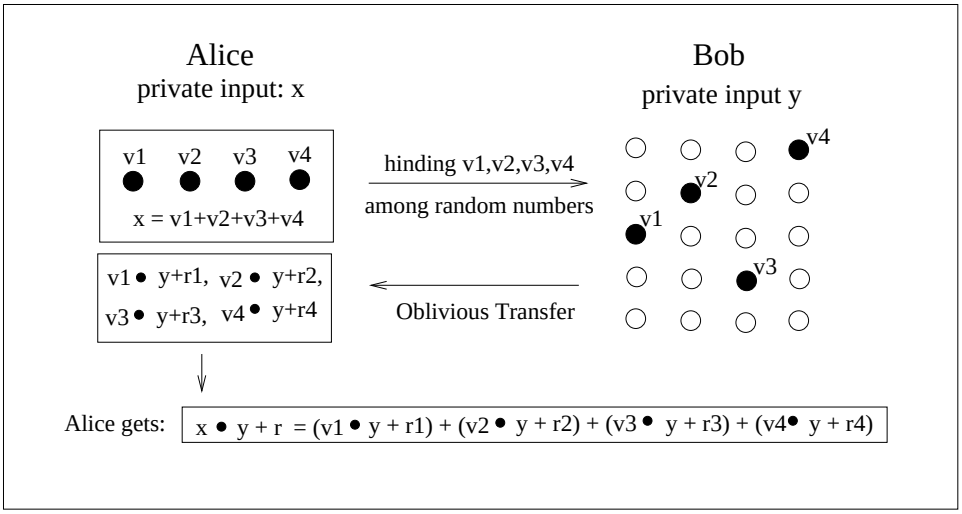

Figure 2: Vector Product Protocol

Protocol

1. Alice and Bob agree on two numbers p and m, such that p^m is so large that conducting p^m additions is computationally infeasible. For example, Alice and Bob could choose p = 2 and m = 1024.
2. Alice generates m random vectors $\vec{u}_1, \ldots, \vec{u}_m$, such that $\vec{x} = \sum_{j=1}^{m} \vec{u}_j$.
3. Bob generates m random numbers $r_1, \ldots, r_m$, such that $b = \sum_{j=1}^{m} r_j$.
4. For each $j = 1, \ldots, m$, Alice and Bob conduct the following sub-steps:
   (a) Alice sends $\vec{h}_1, \ldots, \vec{h}_p$ to Bob, where for a secret k, $\vec{h}_k = \vec{u}_j$ and the rest are random vectors.
   (b) Bob computes $\vec{h}_1 \cdot \vec{y} + r_j, \ldots, \vec{h}_p \cdot \vec{y} + r_j$.
   (c) Alice retrieves the kth element $\vec{h}_k \cdot \vec{y} + r_j = \vec{u}_j \cdot \vec{y} + r_j$ from Bob using oblivious transfer.

5. Alice computes $a = \sum_{j=1}^{m}(\vec{u}_j \cdot \vec{y} + r_j) = \vec{x} \cdot \vec{y} + b$.

Alice preserves her privacy by dividing the vector into m random components each of which is itself hidden using p random vectors, and by privately retrieving the results. Bob is protected by the random numbers r_j which prevent Alice from computing $\vec{y}$ using a linear system of equations.

As in the PIM/Approx/Squ problem Alice and Bob run this protocol for all $t_i \in T$. Alice and Bob then share the results for the scores in the vectors $\vec{a}$ and $\vec{b}$, respectively. This protocol has communication complexity $O(p * m * n * N)$.

5.2 Find Minimum Protocol

The goal is for Alice (but not Bob) to find the minimum element of vector $\vec{s} = (s_1, \ldots, s_N)$ where $\vec{s} = \vec{a} - \vec{b}$, Alice has $\vec{a} = (a_1, \ldots, a_N)$, and Bob has $\vec{b} = (b_1, \ldots, b_N)$. Neither Alice nor Bob should learn order information about the s_i's (e.g., the s_i's sorted order, or even that $s_8 < s_3$). The following protocol is bad in that it illegally reveals such order information between the s_i's to both Alice and Bob, but otherwise it does succeed in letting Alice know the minimum element value s_k, without revealing that s_k to Bob (although Bob does find out k):

Preliminary (bad) protocol for finding the minimum

1. Alice mimics the standard minimum algorithm and, whenever that algorithm requires that s_i be compared to s_j, Alice gives Bob the ordered pair i, j and they do the following:

   (a) Alice computes $u = a_i - a_j = s_i - s_j + b_i - b_j$, and Bob computes $v = b_i - b_j$. (Note that $s_i > s_j$ iff $u < v$.)

   (b) They compare their respective values u and v using Yao's millionaire protocol [34].
   Note. This can be made an asymmetric version of Yao's millionaire problem, in which Alice but not Bob knows the outcome of the comparison, but Bob nevertheless finds out the outcome based on his observation of future pairs of indices that Alice will send him (if $s_i < s_j$ then index i will reappear but not index j).

2. Let k be the index of the minimum. Alice obtains the desired s_k by getting b_k from Bob, e.g. through 1-out-of-N oblivious transfer [26].
   Note. We will explain below how we can prevent Bob from learning k through his observations of the pairs i, j that Alice sends him. Furthermore, oblivious transfer will then no longer be needed to hide k from Bob.

As noted earlier, the above protocol is bad because it reveals too much information during the computation of the minimum. Before running the above protocol Alice and Bob, using a random permutation π that is unknown to both Alice and Bob, could permute the entries of $\vec{a}$, and in the same way permute the entries of $\vec{b}$ (in effect implicitly permuting the entries of $\vec{s}$). But that would not be enough; either one of them could find the secret permutation by comparing the elements of their permuted vector to those of their original vector. In order to prevent this, they have to be also "blinded" by adding a random numbers r_i to each element and a_i and b_i, where r_i is also unknown to both Alice and Bob. The way we achieve the random permutation effect is by using a permutation π that is the composition of two random permutations π_A and π_B, where the former is known to Alice (but not Bob), and the latter is known to Bob (but not Alice). The "additive blinding" of a_i and b_i is by using an $r_i = r'_i + r''_i$ where r'_i is known to Alice (but not Bob), and r''_i is known to Bob (but not Alice). The permutation and blinding are done first by Alice, then again by Bob.

To perform additive blinding, we need a special public key cryptosystem that has the following property: $E_k(x) \cdot E_k(y) = E_k(x+y)$. Such systems are called *homomorphic* cryptosystems and examples include the systems by Benaloh [4], Naccache and Stern [25], Okamoto and Uchiyama [27], and Paillier [28].

Protocol for permuting and additive blinding

Inputs: Alice has $\vec{a}$, Bob has $\vec{b}$, Alice has a random permutation π_A and a random vector $\vec{r'}$ known to her but not to Bob.
Output: Bob obtains the set of N values of the form $b_i + r'_i$ in a permuted order according to π_A (which he does not know). (Note that Alice trivially has the set of N values of the form $a_i + r'_i$ in a permuted order according to π_A, because she knows π_A and $\vec{r'}$).

1. Alice and Bob each generate a key pair for a homomorphic public key system and exchange their public keys. In what follows $E_A(\cdot)$ denotes encryption with Alice's public key, and $D_A(\cdot)$ decryption with Alice's private key (similarly for $E_B(\cdot)$ and $D_B(\cdot)$).

2. Bob encrypts each entry $(b_1, \ldots, b_N)$ using his public key and sends $\vec{b'} = (E_B(b_1), \ldots, E_B(b_N))$ to Alice.

3. Alice does the following:

   (a) She computes $\theta_i = b'_i \cdot E_B(r'_i) = E_B(b_i + r'_i)$, for $i = 1, \ldots, N$.

   (b) She permutes, according to π_A, the order of the θ_i's: Let $\vec{b''}$ denote the vector of permuted θ_i's.

   (c) Alice sends $\vec{b''}$ to Bob.

4. Bob decrypts the entries of $\vec{b''}$, obtaining the set of N values of the form $b_i + r'_i$ in a permuted order according to π_A (which he does not know). Note that Alice trivially has the set of N values of the form $a_i + r'_i$ in a permuted order according to π_A (which she does know).

Suppose Alice and Bob run the above "permute and blind" protocol once, and then follow it with the earlier-given preliminary ("bad") "find minimum" protocol in which the permuted-and-blinded data is used instead of the original vectors $\vec{a}$ and $\vec{b}$. Unlike before, Bob now learns nothing of the relative orderings of the s_i's, and does not learn the index k for which s_k is minimum. But Alice still learns too much (because she knows both π_A and $\vec{r'}$). This is easily fixed: After running the above "permute and blind" protocol, we run it again, on the already permuted-and-blinded data, but with roles of Alice and Bob interchanged. As a result, Alice and Bob end up with doubly-permuted and doubly-blinded data: Permuted according to π_A followed by π_B, the composition of which is unknown to both of them (because Alice knows only π_A and Bob knows only π_B), and additively blinded according to a random vector that is unknown to both of them because it is the sum of two random vectors the first of which is known only to Alice and the second only to Bob. In other words, Bob now has a random permutation π of the set of N values $b_i + \hat{r}_i$, and Alice has the same permutation π of the set of N values $a_i + \hat{r}_i$, and neither Alice nor Bob know π or $\hat{r}_i$. It is finally safe to use the earlier-given (and no longer bad!) preliminary protocol.

This protocol has communication cost $O(N) * \lambda + O(N) * \mu$, where λ is the cost of encrypting a vector element using homomorphic ciphers and μ is the cost of performing a Yao's millionaire protocol on a vector element. Also, because we use public-key cryptosystems, probabilistic encryption for vector elements whose domain could be exhaustively searched, might be needed, which further increases the communication cost for the homomorphic encryption.

6 Conclusion and Future Work

We have developed four models for secure remote database access, and presented a class of problems and solutions for these models. For some problems, such as SSO/Approx/Squ and SSCO/Approx/Squ problems, our solutions are practical, and they only need $O(n)$ and $O(n^2)$ communication cost, respectively; while for PIM/Approx and PIMPD/Approx problems, our results are still at the theoretical stage because of their high communication cost. Improving the communication cost for those solutions is one avenue for future work: We suspect that, whenever there is a dependence on N, that dependence could be made sub-linear (perhaps logarithmic) by combining our methods with the known powerful higher dimensional indexing techniques [31, 7, 2, 19, 32, 15]. However, combining those schemes with our protocols will not be a trivial task, and the increase in the constant factors hiding behind the "big-oh" notation may well negate the benefits of the asymptotic sub-linearity in N; for example, in a tree search for processing the query, Bob has to be prevented from knowing what nodes of his tree are visited when processing the query (otherwise he gets information about the query), which requires using a PIR-like protocol at each node down the tree. But even that is not enough: Alice herself must be prevented from learning anything about Bob's data other than the answer to her query, but in most of the tree-based schemes in the literature the comparison at a node of the search tree gives information about the data that is associated with that node (these schemes were designed for an environment where the searcher is the owner, and may require substantial modification before they are used in our context).

Another avenue for future work is the pattern matching of branching structures: The pattern matching problems that we have discussed only involve patterns of simple linear structure; in many applications, patterns have a branching structure, such as a tree or a DAG. The M/Approx/Edit/Tree problem in our model is one of the examples. Developing a secure protocol to deal with this type of query is a challenging problem.

References

[1] A. Apostolico and Z. Galil, editors. *Pattern Matching Algorithms*. Oxford University Press, 1997.

[2] S. Arya. Ph.d thesis: Nearest neighbor searching and applications. Technical Report CS-TR-3490, University of Maryland at College Park, June 1995.

[3] M. Atallah and J. Rice. Secure outsourcing of scientific computations. Technical Report COAST TR 98-15, Department of Computer Science, Purdue University, 1998.

[4] J. Benaloh. Dense probabilistic encryption. In *Proceedings of the Workshop on Selected Areas of Cryptography*, pages 120–128, Kingston, ON, May 1994.

[5] B. Chor and N. Gilboa. Computationally private information retrieval (extended abstract). In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, El Paso, TX USA, May 4-6 1997.

[6] M. Crochemore and W. Rytter. *Text Algorithms*. Oxford University Press, 1994.

[7] R. Agrawal, C. Faloutsos and A. Swami. Efficient similarity search in sequence databases. In *Proceeding of the Fourth Int'l Conference on Foundations of Data Organization and Algorithms*, October 1993. Also in Lecture Notes in Computer Science 730, Springer Verlag, 1993, 69–84.

[8] B. Chor, N. Gilboa and M. Naor. Private information retrieval by keywords. Technical Report TR CS0917, Department of Computer Science, Technion, 1997.

[9] O. Goldreich. Secure multi-party computation (working draft). Available from http://www.wisdom.weizmann.ac.il/home/oded/public_html/foc.html, 1998.

[10] P. F. Syverson, D. M. Goldschlag and M. G. Reed. Anonymous connections and onion routing. In *Proceedings of 1997 IEEE Symposium on Security and Privacy*, Oakland, California, USA, May 5-7 1997.

[11] S. Goldwasser. Multi-party computations: Past and present. In *Proceedings of the sixteenth annual ACM symposium on Principles of distributed computing*, Santa Barbara, CA USA, August 21-24 1997.

[12] Y. Gertner, S. Goldwasser and T. Malkin. A random server model for private information retrieval. In *2nd International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM '98)*, 1998.

[13] R. Gonzalezi and R. Woods. *Digital Image Processing*. Addison-Wesley, Reading, MA, 1992.

[14] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Comutational Biology*. Cambridge University Press, 1997.

[15] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *ACM SIGMOD Workshop on Data Mining and Knowledge Discovery*, pages 163–174, Boston, MA, 1984.

[16] G. Di-Crescenzo, Y. Ishai and R. Ostrovsky. Universal service-providers for database private information retrieval. In *Proceedings of the 17th Annual ACM Symposium on Principles of Distributed Computing*, September 21 1998.

[17] Y. Ishai and E. Kushilevitz. Improved upper bounds on information-theoretic private information retrieval (extended abstract). In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, Atlanta, GA USA, May 1-4 1999.

[18] A. Jain. *Fundamentals of Digital Image Processing*. Prentice Hall, Englewood Cliffs, NJ, 1989.

[19] J. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, 1997.

[20] B. Chor, O. Goldreich, E. Kushilevitz and M. Sudan. Private information retrieval. In *Proceedings of IEEE Symposium on Foundations of Computer Science*, Milwaukee, WI USA, October 23-25 1995.

[21] E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Proceedings of the 38th annual IEEE computer society conference on Foundation of Computer Science*, Miami Beach, Florida USA, October 20-22 1997.

[22] Y. Gertner, Y. Ishai, E. Kushilevitz and T. Malkin. Protecting data privacy in private information retrieval schemes. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, Dallas, TX USA, May 24-26 1998.

[23] C. Cachin, S. Micali and M. Stadler. Computationally private information retrieval with polylogarithmic communication. *Advances in Cryptology: EUROCRYPT '99, Lecture Notes in Computer Science*, 1592:402–414, 1999.

[24] O. Goldreich, S. Micali and A. Wigderson. How to play any mental game. In *Proceedings of the 19th annual ACM symposium on Theory of computing*, pages 218–229, 1987.

[25] D. Naccache and J. Stern. A new cryptosystem based on higher residues. In *Proceedings of the 5th ACM Conference on Computer and Communications Security*, pages 59–66, 1998.

[26] M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation (extended abstract). In *Proceedings of the 31th ACM Symposium on Theory of Computing*, pages 245–254, Atanta, GA, USA, May 1-4 1999.

[27] T. Okamoto and S. Uchiyama. An efficient public-key cryptosystem. In *Advances in Cryptology – EUROCRYPT 98*, pages 308–318, 1998.

[28] P. Paillier. Public-key cryptosystems based on composite degree residue classes. In *Advances in Cryptology – EUROCRYPT 99*, pages 223–238, 1999.

[29] M. Rabin. How to exchange secrets by oblivious transfer. Technical Report Tech. Memo TR-81, Aiken Computation Laboratory, 1981.

[30] M. K. Reiter and A. D. Rubin. Crowds: anonymity for web transaction. *ACM Transactions on Information and System Security*, 1(1):Pages 66–92, 1998.

[31] Hanan Samet. Multidimensional data structures. In Mikhail J. Atallah, editor, *Algorithms and Theory of Computation Handbook*, chapter 18. CRC Press, 1999.

[32] N. Beckmann, H-P. Kriegel, R. Schneider, B. Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *ACM SIGMOD Workshop on Data Mining and Knowledge Discovery*, pages 322–331, Atlantic City, NJ, 1990.

[33] D. Song, D. Wagner and A. Perrig. Practical techniques for searches on encrypted data. In *Proceedings of 2000 IEEE Symposium on Security and Privacy*, Oakland, California, USA, May 14-17 2000.

[34] A. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science*, 1982.