CERIAS Tech Report 2005-126

Secure Outsourcing of Sequence Comparisons

by Mikhail J. Atallah, Jiangtao Li

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

Int J Inf Secur (2005) 00: 1–11 / Digital Object Identifier (DOI) 10.1007/s10207-005-0070-3

Secure outsourcing of sequence comparisons*

Mikhail J. Atallah, Jiangtao Li

CERIAS and Department of Computer Sciences, Purdue University, 250 N. University Street, West Lafayette, IN 47907, USA
e-mail: {mja,jtli}@cs.purdue.edu

Published online: ■■ 2005 –

Abstract. Internet computing technologies, like grid computing, enable a weak computational device connected to such a grid to be less limited by its inadequate local computational, storage, and bandwidth resources. However, such a weak computational device (PDA, smartcard, sensor, etc.) often cannot avail itself of the abundant resources available on the network because its data are sensitive. This motivates the design of techniques for computational outsourcing in a privacy-preserving manner, i.e., without revealing to the remote agents whose computational power is being used either one's data or the outcome of the computation. This paper investigates such secure outsourcing for widely applicable sequence comparison problems and gives an efficient protocol for a customer to securely outsource sequence comparisons to two remote agents. The local computations done by the customer are linear in the size of the sequences, and the computational cost and amount of communication done by the external agents are close to the time complexity of the best known algorithm for solving the problem on a single machine.

Keywords: Privacy – Secure outsourcing – Secure multiparty computation – Sequence comparison – Edit distance

* Portions of this work were supported by Grants IIS-0325345, IIS-0219560, IIS-0312357, and IIS-0242421 from the National Science Foundation, Contract N00014-02-1-0364 from the Office of Naval Research, by sponsors of the Center for Education and Research in Information Assurance and Security, and by Purdue Discovery Park's e-enterprise Center. A preliminary version [3] of this paper was presented at the 4th Workshop on Privacy Enhancing Technologies, May 2004, Toronto, Canada.

1 Introduction

Large-scale problems in the physical and life sciences are being revolutionized by Internet computing technologies, like grid computing [10], that make possible the massive cooperative sharing of computational power, bandwidth, storage, and data. A weak computational device, once connected to such a grid, is no longer limited by its slow speed, small amounts of local storage, and limited bandwidth: it can avail itself of the abundance of these resources that is available elsewhere on the network. An impediment to the use of "computational outsourcing" is that the data in question are often sensitive, e.g., of national security importance, or proprietary and containing commercial secrets, or to be kept private for legal requirements such as the HIPAA legislation, Gramm-Leach-Bliley, or similar laws. A prime example of this is DNA sequence comparisons: they are expensive enough to warrant remotely using the computing power available at powerful remote servers and supercomputers, yet sensitive enough to give pause to anyone concerned that some unscrupulous person at the remote site may leak the DNA sequences or the comparison's outcome, or may subject the DNA to a battery of unauthorized tests whose outcome could have such grave consequences as jeopardizing an individual's insurability, employability, etc. Techniques for outsourcing expensive computational tasks in a privacy-preserving manner are therefore an important research goal. This paper is a step in this direction in that it gives a protocol for the secure outsourcing of the most important sequence comparison computation: the "string editing" problem, i.e., computing the edit distance between two strings. The edit distance is one of the most widely used notions of similarity: it is the least-cost set of insertions, deletions, and substitutions required to transform one string into another. Essentially the same proto-

MS ID: IJIS0070
31 January 2005 16:04 CET

col can solve the larger class of comparisons whose standard dynamic programming solution is similar in structure to that of string editing. The generalizations of edit distance that are solved by the same kind of dynamic programming recurrence relation as the one for edit distance cover an even wider domain of applications. We use string editing here merely as the prototypical solution for this general class of dynamic programming recurrences.

In various ways and forms, sequence comparisons arise in many applications other than molecular sequence comparison, notably, in text editing, speech recognition, machine vision, etc. In fact, the dynamic programming solution to this problem was independently discovered by no fewer than 14 different researchers [27] and is given a different name by each discipline where it was independently discovered (Needleman-Wunsch by biologists, Wagner-Fischer by computer scientists, etc). For this reason, these problems have been studied rather extensively in the past and form the object of several papers [16, 17, 21, 26, 27, 29, 32], to list a few). The problems are typically solved by a serial algorithm in $\Theta(mn)$ time and space through dynamic programming (cf., for example, [32]). When huge sequences are involved, the quadratic time complexity of the problem quickly becomes prohibitively expensive, requiring considerable power. Such supercomputing power is widely available, but sending the data to such remote agents is problematic if the sequence data are sensitive, the outcome of the comparison is to be kept private, or both. In such cases, one can make a case for a technology that makes it possible for the customer to have the problem solved remotely but without revealing to the remote supercomputing sites either the inputs to the computation or its outcome.

In other words, we assume that Carol has two private sequences, λ and μ, and wants to compute the similarity between these two sequences. Carol only has a weak computational device that is incapable of performing the sequence comparison locally. In order to get the result, Carol has to outsource the computation task to some external entities, the agents. If Carol trusted the agents, she could send the sequences directly to the external agents and ask them to compute the similarity on her behalf. However, if Carol is concerned about privacy, it is not acceptable to send the sequences to external agents because this would reveal too much information to these agents – both the sequences and the result. Our result is a protocol that computes the similarity of the sequences yet inherently safeguards the privacy of Carol's data. Assuming the two external agents do not conspire with each other against Carol by sharing the data that she sends to them, they learn nothing about the actual data and actual result.

The dynamic programming recurrence relation that subtends the solution to this problem also serves to solve many other important related problems (either as special cases or as generalizations that have the same dynamic programming kind of solution). These include the longest common subsequence problem and the problem of approximate matching between a pattern sequence and text sequence (there is a huge literature of published work for the notion of approximate pattern matching and its connection to the sequence alignment problem). Any solution to the general sequence comparison problem could also be used to solve these related problems. For example, our protocol can enable a weak PDA to securely outsource the computation of the `Unix` command

`diff` *file1 file2* | `wc`

to two agents where the agents learn nothing about *file1*, *file2*, and the result.

We now more precisely state the edit distance problem, in which the cost of an insertion or deletion or substitution is a symbol-dependent nonnegative weight, and the edit distance is then the *least-cost* set of insertions, deletions, and substitutions required to transform one string into another. More formally, if we let λ be a string of length n, $\lambda = \lambda_1 \dots \lambda_n$, and μ be a string of length m, $\mu = \mu_1 \dots \mu_m$, both over some alphabet Σ. There are three types of allowed *edit operations* to be done on λ: insertion of a symbol, deletion of a symbol, and substitution of one symbol by another. Each operation has a cost associated with it, namely, $I(a)$ denotes the cost of inserting the symbol a, $D(a)$ denotes the cost of deleting a, and $S(a, b)$ denotes the cost of substituting a with b. Each sequence of operations that transforms λ into μ has a *cost* associated with it (which is equal to the sum of the costs of the operations in it), and the least-cost of such sequence is the *edit distance*. The *edit path* is the actual sequence of operations that corresponds to the edit distance. Our outsourcing solution allows arbitrary $I(a)$, $D(b)$, and $S(a, b)$ values, and we give better solutions for two special cases: (i) $S(a, b) = |a - b|$ and (ii) unit insertion/deletion cost and $S(a, b) = 0$ if $a = b$ and $S(a, b) = +\infty$ if $a \neq b$ (in effect forbidding substitutions).

The rest of paper is organized as follows. We begin with a brief introduction of previous work in Sect. 2. Then we describe some building blocks in Sect. 3. In Sect. 4, we present the secure outsourcing protocol for computing string edit distance. Section 5 extends the protocol so as to compute the edit path. Section 6 concludes the paper.

2 Related work

Recently, Atallah et al. [2] developed an efficient protocol for sequence comparisons in the secure two-party computation framework in which each party has a private string; the protocol enables two parties to compute the edit distance of two sequences such that neither party learns anything about the private sequence of the other party. They [2] use dynamic programming to compare sequences, but in an additively split way – each party maintains a matrix, the summation of two matrices is the

real matrix implicitly used to compute edit distance. Our protocol directly builds on their work, but is also quite different and more difficult in the following ways:

- We can no longer afford to have the customer carry out quadratic work or communication: whereas in [2] there was "balance" in that all participants had equal computational and communication power, in our case the participant to whom all of the data and answer belong is asymmetrically weaker and is limited to a *linear* amount of computation and communication (hence cannot directly participate or help in each step of the quadratic-complexity dynamic programming solution).
- An even more crucial difference is the special difficulty this paper's framework faces in dealing with the costs table, that is, the table that contains the costs of deleting a symbol, inserting a symbol, and substituting one symbol for another: there is a quadratic number of accesses to this table, and the external agents cannot be allowed to learn which entry of the table is being consulted (because that would leak information about the inputs), yet the input owner's help cannot be enlisted for such table accesses because there is a quadratic number of them (recall that the owner is limited to linear work and communication – which is unavoidable).

Secure outsourcing of sequence comparisons adds to a growing list of problems considered in this framework (e.g. [4, 5, 12, 14, 18, 24], and others). We briefly review these next. In the server-aided secret computation literature (e.g., [5, 12, 14, 18, 24], to list a few), a weak smartcard performs public-key encryptions by "borrowing" computing power from an untrusted server, without revealing to that server its private information. These papers deal primarily with the important problem of modular exponentiations. The paper by [4] deals primarily with outsourcing of scientific computations.

Boneh et al. [6] recently proposed a new public-key encryption system with which a mail server can perform keyword search on encrypted data with the help of the public-key holder. Ogata and Kurosawa [22] introduced the notion of oblivious keyword search. In an oblivious keyword search protocol, Alice has a keyword W and conducts keyword search on Bob's database; in the end, Alice obtains a set of data that includes W while Bob learns nothing about W. Ogata and Kurosawa [22] proposed a couple of efficient oblivious keyword search protocols. These keyword search schemes [6, 22] could be viewed as other special cases of secure outsourcing where the client outsources the keyword search operations to the server.

In the the privacy homomorphism approach proposed in [25], the outsourcing agent is used as a permanent repository of data, performing certain operations on it and maintaining certain predicates, whereas the customer needs only to decrypt the data from the agent to obtain the real data; the secure outsourcing framework differs in that the customer is not interested in keeping data permanently with the external agents; instead, the customer only wants to temporarily use their superior computational power.

Du and Atallah have developed several models for secure remote database access with approximate matching [8]. One of the models that is related to our work is the secure storage outsourcing model where a customer who lacks storage space outsources her database to an external agent. The customer needs to query her database from time to time without revealing to the agent the queries and the results. Several protocols for other distance metrics were given, including Hamming distance and the L_1 and L_2 distance metrics. All these metrics considered in [8] were between strings that have *the same length* – it is indeed a limitation of the techniques in [8] that they do not extend to the present situation where the strings are of different length and insertions and deletions are part of the definition. This makes the problem substantially different, as the edit distance algorithm is described by a dynamic program that computes it, rather than as a simple one-line mathematical expression to be securely computed.

3 Preliminaries

Giving the full-fledged protocol would make it too long and rather hard to comprehend. This section aims at making the later presentation of the protocol much crisper by presenting some of the ideas and building blocks for it ahead of time, right after a brief review of the standard dynamic programming solution to string edit.

3.1 Review of edit distance via dynamic programming

We first briefly review the standard dynamic programming algorithm for computing edit distance. Let $M(i,j)$, $(0 \le i \le n,\ 0 \le j \le m)$ be the minimum cost of transforming the prefix of λ of length i into the prefix of μ of length j, i.e., of transforming $\lambda_1 \ldots \lambda_i$ into $\mu_1 \ldots \mu_j$. Then $M(0,0) = 0$, $M(0,j) = \sum_{k=1}^{j} I(\mu_j)$ for $1 \le j \le m$, $M(i,0) = \sum_{k=1}^{i} D(\lambda_i)$ for $1 \le i \le n$, and for positive i and j we have

$$M(i,j) = \min \begin{cases} M(i-1,j-1) + S(\lambda_i, \mu_j) \\ M(i-1,j) + D(\lambda_i) \\ M(i,j-1) + I(\mu_j) \end{cases}$$

for all i, j, $1 \le i \le n$ and $1 \le j \le m$. Hence $M(i,j)$ can be evaluated row by row or column by column in $\Theta(mn)$ time [32]. Observe that, of all entries of the M-matrix, only the three entries $M(i-1,j-1)$, $M(i-1,j)$, and $M(i,j-1)$ are involved in the computation of the final value of $M(i,j)$.

Not only does the above dynamic program for computing M depend on both λ and μ, but even if M could be computed without knowing λ and μ, the problem remains that M itself is too revealing: it reveals not only the overall

edit distance, but also the edit distance from every prefix of λ to every prefix of μ. It is required in our problem that the external agents learn nothing about the actual sequences and the results. The M-matrix should therefore not be known to the agents. It can of course not be stored at the customer's site, as it is a requirement that the customer be limited to $O(m+n)$ time and storage space.

3.2 Framework

We use two noncolluding agents in our protocol. Both the input sequences (λ and μ) and the intermediate results (the M-matrix) are additively split between the two agents in such a way that neither of the agents learns anything about the real inputs and results, but the two agents together can implicitly use the M-matrix without knowing it, that is, obtaining additively split answers "as if" they knew M. They have to do so without the help of the customer, as the customer is incapable of quadratic computation time or storage space. More details on how this is done are given below.

In the rest of the paper, we use the following notations. We use $\mathcal{C}$ to denote the customer, $\mathcal{A}_1$ the first agent, and $\mathcal{A}_2$ the second agent. Any items superscripted with $'$ are known to $\mathcal{A}_1$but not to $\mathcal{A}_2$, and those superscripted with $''$ are known to $\mathcal{A}_2$ but not to $\mathcal{A}_1$. In what follows, we often *additively split* an item x between the two agents $\mathcal{A}_1$ and $\mathcal{A}_2$, i.e., we assume that $\mathcal{A}_1$ has an x' and $\mathcal{A}_2$ has an x'' such that $x = x' + x''$; we do this splitting for the purpose of hiding x from either agent. If arithmetic is modular, then this kind of additive splitting of x hides it, in an information-theoretic sense, from $\mathcal{A}_1$ and $\mathcal{A}_2$. If, however, arithmetic is not modular, then even when x' and x'' can be negative and are very large compared to x, the "hiding" of x is valid in a practical but not in an information-theoretic sense.

3.2.1 Splitting λ and μ

Let λ and μ be two sequences over some finite alphabet $\Sigma = \{0, \ldots, \sigma-1\}$. This could be a known fixed set of symbols (e.g., in biology $\Sigma = \{A, C, T, G\}$) or the domain of a hash function that maps a potentially infinite alphabet into a finite domain. $\mathcal{C}$ splits λ into λ' and λ'' such that λ' and λ'' are over the same alphabet Σ and their sum is λ, i.e., $\lambda_i = \lambda'_i + \lambda''_i \bmod \sigma$ for all $1 \le i \le n$. To split λ, $\mathcal{C}$ can first generate a random sequence λ' of length n, then set $\lambda''_i = \lambda_i - \lambda'_i \bmod \sigma$ for all $1 \le i \le n$. Similarly, $\mathcal{C}$ splits μ into μ' and μ'' such that $\mu_i = \mu'_i + \mu''_i \bmod \sigma$ for all $1 \le i \le m$. In the edit distance protocol, $\mathcal{C}$ sends λ' and μ' to $\mathcal{A}_1$ and sends λ'' and μ'' to $\mathcal{A}_2$.

3.2.2 Splitting M

Our edit distance protocol computes the same matrix as the dynamic programming algorithm in the same order (e.g., row by row). Like [2], the matrix M in our protocol is additively shared between $\mathcal{A}_1$ and $\mathcal{A}_2$: $\mathcal{A}_1$ and $\mathcal{A}_2$ each hold a matrix M' and M'', respectively, the sum of which is the matrix M, i.e., $M = M' + M''$; the protocol will maintain this property as an invariant through all its steps. The main challenge in our protocol is that the comparands and outcome of each comparison, as well as the indices of the minimum elements, have to be shared (in the sense that neither party individually knows them).

3.2.3 Hiding the sequence lengths

Splitting a sequence effectively hides its content but fails to hide its length. In some situations, even the lengths of the sequences are sensitive and must be hidden or, at least, somewhat obfuscated. We now briefly sketch how to pad the sequences and obtain new, longer sequences whose edit distance is the same as that between the original sentences. Let $\hat{m}$ and $\hat{n}$ be the respective new lengths (with padding); assume that randomly choosing $\hat{m}$ from the interval $[m, 2m]$ provides enough obfuscation of m, and similarly $\hat{n}$ from the interval $[n, 2n]$.

We introduce a new special symbol "\$" to the alphabet Σ such that the cost of insertion and deletion of this symbol is 0 (i.e., $I(\$) = D(\$) = 0$), and the cost of substitution of this symbol is infinity (i.e., $S(\$, a) = S(a, \$) = +\infty$ for every symbol a in Σ). The customer appends "\$"s to the end of λ and μ to turn their respective lengths into the target values $\hat{n}$ and $\hat{m}$ before splitting and sending them to the agents. This padding has following two properties: (1) the edit distance between the padded sequences is the same as the edit distance between the original sequences and (2) the agents cannot figure out how many "\$"s were padded into a sequence because of the random split of the sequence.

To avoid unnecessarily cluttering the exposition, we assume λ and μ are already padded with "\$"s before the protocol; thus we assume the lengths of λ and μ are still n and m, respectively, and the alphabet Σ is still $\{0, \ldots, \sigma-1\}$.

3.3 Secure table lookup protocol for split data

Recall that the $\sigma \times \sigma$ size cost table S is public, hence known to both $\mathcal{A}_1$ and $\mathcal{A}_2$; we make no assumptions about the costs in the table (they can be arbitrary, not necessarily between 0 and $\sigma-1$). Recall that $\mathcal{A}_1$ and $\mathcal{A}_2$ share additively each symbol α from λ and β from μ, i.e., $\alpha = \alpha' + \alpha'' \bmod \sigma$ and $\beta = \beta' + \beta'' \bmod \sigma$, where $\mathcal{A}_1$ has α' and β' and $\mathcal{A}_2$ has α'' and β''. $\mathcal{A}_1$ and $\mathcal{A}_2$ want to cooperatively look up the value $S(\alpha, \beta)$ from the cost table S, but without either of them knowing which entry of S was accessed and what value was returned by the access (so that value itself must be additively split). The protocol below solves this lookup problem in one round and $O(\sigma^2)$ computation and communication; note that naively using the protocol below

$O(mn)$ times would result in an $O(\sigma^2 mn)$ computation and communication complexity for the overall sequence comparison problem, not the $O(\sigma mn)$ performance we claim (and that will be substantiated later in the paper).

Protocol 1. *Secure table lookup protocol*

Input $\mathcal{A}_1$ has α' and β' and $\mathcal{A}_2$ has α'' and β'' such that $\alpha = \alpha' + \alpha'' \bmod \sigma$ and $\beta = \beta' + \beta'' \bmod \sigma$.

Output $\mathcal{A}_1$ obtains a number a, and $\mathcal{A}_2$ obtains a number b, such that $a + b = S(\alpha, \beta)$.

The protocol steps are:

1. $\mathcal{A}_1$ generates a key pair for a homomorphic semantically secure public-key system and sends the public key to $\mathcal{A}_2$ (any of the existing systems will do, e.g., [19, 23]). In what follows $E(\cdot)$ denotes encryption with $\mathcal{A}_1$'s public key and $E^{-1}(\cdot)$ decryption with $\mathcal{A}_1$'s private key. (Recall that the homomorphic property implies that $E(x) * E(y) = E(x+y)$ and semantic security implies that $E(x)$ reveals nothing about x, so that $x = y$ need not imply $E(x) = E(y)$.)
2. $\mathcal{A}_1$ generates a $\sigma \times \sigma$ size table $\hat{S}$ with entry $\hat{S}(i,j)$ equal to $E(S(i + \alpha' \bmod \sigma, j + \beta' \bmod \sigma))$ for all $0 \leq i, j \leq \sigma - 1$ and sends that table $\hat{S}$ to $\mathcal{A}_2$.
3. $\mathcal{A}_2$ picks up the (α'', β'')th entry from the table received in the previous step, which is $\hat{S}(\alpha'', \beta'') = E(S(\alpha, \beta))$. $\mathcal{A}_2$ then generates a random number b, then computes $\theta = E(S(\alpha, \beta)) * E(-b) = E(S(\alpha, \beta) - b)$, and sends it back to $\mathcal{A}_1$.
4. $\mathcal{A}_1$ decrypts the value received from $\mathcal{A}_2$ and gets $a = E^{-1}(E(S(\alpha, \beta) - b)) = S(\alpha, \beta) - b$.

As required, $a + b = S(\alpha, \beta)$, and $\mathcal{A}_1$ and $\mathcal{A}_2$ do not learn anything about the other party from the protocol. The computation and communication cost of this protocol is $O(\sigma^2)$. Note that the multiplicative constant implicit in the "$O(\sigma^2)$" notion is fairly large due to the homomorphic encryptions, i.e., this protocol requires $O(\sigma^2)$ homomorphic encryptions for $\mathcal{A}_1$ and $O(1)$ homomorphic encryptions for $\mathcal{A}_2$. The communication cost of this protocol is around $c\sigma^2$, where c is the length of the homomorphic encryption.

4 Edit distance protocol

We now "put the pieces together" and give the overall protocol. We begin with the general case of arbitrary $I(a)$, $D(b)$, $S(a,b)$. Then two special cases are considered. One is the case of arbitrary $I(a)$ and $D(b)$, but $S(a,b) = |a - b|$. The other is the practical case of unit insertion/deletion cost and forbidden substitutions (i.e., $S(a,b)$ is 0 if $a = b$ and $+\infty$ otherwise). For all the above cases, the cost of computation and communication by the customer is linear to the size of the input. The cost of computation and communication by agents is $O(\sigma mn)$ for the general case and $O(mn)$ for the two special cases.

4.1 The general case: arbitrary $I(a)$, $D(b)$, $S(a,b)$

In this section, we begin with a preliminary solution that is not our best but serves as a useful "warmup" to the more efficient solution that comes later in this section.

4.1.1 A preliminary version of the protocol

Recall that $\mathcal{C}$ splits λ into λ' and λ'' and μ into μ' and μ'', then sends λ' and μ' to $\mathcal{A}_1$, and sends λ'' and μ'' to $\mathcal{A}_2$. $\mathcal{A}_1$ and $\mathcal{A}_2$ each maintain a matrix M' and (respectively) M'' such that $M = M' + M''$. $\mathcal{A}_1$ and $\mathcal{A}_2$ compute each element $M(i,j)$ in an additively split fashion; this is done as prescribed in the recursive edit distance formula by $\mathcal{A}_1$ and $\mathcal{A}_2$ updating their respective M' and M''. After doing so, $\mathcal{A}_1$ and $\mathcal{A}_2$ send their respective $M'(n,m)$ and $M''(n,m)$ back to $\mathcal{C}$. $\mathcal{C}$ can then obtain the edit distance $M(n,m) = M'(n,m) + M''(n,m)$.

During the computation of each element $M(i,j)$, $S(\lambda_i, \mu_j)$ has to be computed by $\mathcal{A}_1$ and $\mathcal{A}_2$ in an additively split fashion and without the help of $\mathcal{C}$, which implies that the substitution table S should be known by both $\mathcal{A}_1$ and $\mathcal{A}_2$. Hence, $\mathcal{C}$ needs to send the table to both of the agents during the initialization phase of the protocol. The content of the table is not private and need not be disguised.

Initialization of matrices

M' and M'' should be initialized so that their sum M has $M(0,j)$ and $M(i,0)$ equal to the values specified in Sect. 3.1. The $M(i,j)$ entries for nonzero i and j can be random (they will be computed later, after the initialization). The following initializes the M' and M'' matrices:

1. $\mathcal{C}$ generates two vectors of random numbers $\mathbf{a} = (a_1, \ldots, a_n)$ and $\mathbf{b} = (b_1, \ldots, b_m)$. Then $\mathcal{C}$ computes two vectors $\mathbf{c} = (c_1, \ldots, c_n)$ and $\mathbf{d} = (d_1, \ldots, d_m)$, where
   (a) $c_i = \sum_{k=1}^{i} D(\lambda_k) - a_i$ for $1 \leq i \leq n$,
   (b) $d_j = \sum_{k=1}^{j} I(\mu_k) - b_j$ for $1 \leq j \leq m$.
   $\mathcal{C}$ sends to $\mathcal{A}_1$ the vectors $\mathbf{b}, \mathbf{c}$ and to $\mathcal{A}_2$ the vectors $\mathbf{a}, \mathbf{d}$.
2. $\mathcal{A}_1$ sets $M'(0,j) = b_j$ for $1 \leq j \leq m$ and sets $M'(i,0) = c_i$ for $1 \leq i \leq n$. All the other entries of M' are set to 0.
3. $\mathcal{A}_2$ sets $M''(i,0) = a_i$ for $1 \leq i \leq n$ and sets $M''(0,j) = d_j$ for $1 \leq j \leq m$. All the other entries of M'' are set to 0.

Note that the above implicitly initializes $M(i,j)$ in the correct way because it results in

- $M'(0,0) + M''(0,0) = 0$;
- $M'(0,j) + M''(0,j) = \sum_{k=1}^{j} I(\mu_k)$ for $1 \leq j \leq m$;
- $M'(i,0) + M''(i,0) = \sum_{k=1}^{i} D(\lambda_k)$ for $1 \leq i \leq n$.

Neither $\mathcal{A}_1$ nor $\mathcal{A}_2$ gains any information about λ and μ from the initialization of their matrices because the two vectors they each receive from $\mathcal{C}$ look random to them.

Mimicking a step of the dynamic program

The following protocol describes how an $M(i,j)$ computation is done by $\mathcal{A}_1$ and $\mathcal{A}_2$, i.e., how they modify their respective $M'(i,j)$ and $M''(i,j)$, thus implicitly computing the final $M(i,j)$ without either of them learning which update was performed.

1. $\mathcal{A}_1$ and $\mathcal{A}_2$ use the secure table lookup protocol with inputs λ'_i and μ'_j from $\mathcal{A}_1$ and inputs λ''_i and μ''_j from $\mathcal{A}_2$. As a result, $\mathcal{A}_1$ obtains γ' and $\mathcal{A}_2$ obtains γ'' such that

$$\begin{aligned}\gamma' + \gamma'' &= S(\lambda'_i + \lambda''_i \bmod \sigma, \mu'_j + \mu''_j \bmod \sigma)\\ &= S(\lambda_i, \mu_j).\end{aligned}$$

   $\mathcal{A}_1$then forms $u' = M'(i-1,j-1) + \gamma'$ and Bob forms $u'' = M''(i-1,j-1) + \gamma''$. Observe that $u' + u'' = M(i-1,j-1) + S(\lambda_i, \mu_j)$, which is one of the three quantities involved in the update step for $M(i,j)$ in the dynamic program.
2. $\mathcal{A}_1$ computes $v' = M'(i-1,j) + M'(i,0) - M'(i-1,0) = M'(i-1,0) + D(\lambda_i) - a_i + a_{i-1}$, and $\mathcal{A}_2$ computes $v'' = M''(i-1,j) + M''(i,0) - M''(i-1,0) = M''(i-1,j) + a_i - a_{i-1}$. Observe that $u_A + u_B = M(i-1,j) + D(\lambda_i)$, which is one of the three quantities involved in the update step for $M(i,j)$ in the dynamic program.
3. $\mathcal{A}_1$ computes $w' = M'(i,j-1) + M'(0,j) - M'(0,j-1) = M'(i,j-1) + b_j - b_{j-1}$, and $\mathcal{A}_2$ computes $w'' = M''(i,j-1) + M''(0,j) - M''(0,j-1) = M''(i,j-1)$
   $+I(\mu_j) - b_j + b_{j-1}$. Observe that $w' + w'' = M(i,j-1) + D(\mu_j)$, which is one of the three quantities involved in the update step for $M(i,j)$ in the dynamic program.
4. $\mathcal{A}_1$ and $\mathcal{A}_2$ use the minimum finding protocol for split data (described in [2]) on their respective vectors (u', v', w') and (u'', v'', w''). As a result, $\mathcal{A}_1$ gets an x' and $\mathcal{A}_2$ gets an x'' whose sum $x' + x''$ is

$$\begin{aligned}&\min(u' + u'', v' + v'', w' + w'')\\ &= \min \begin{pmatrix} M(i-1,j-1) + S(\lambda_i, \mu_j) \\ M(i-1,j) + D(\lambda_i) \\ M(i,j-1) + I(\mu_j) \end{pmatrix}.\end{aligned}$$

5. $\mathcal{A}_1$ sets $M'(i,j)$ equal to x', and $\mathcal{A}_2$ sets $M''(i,j)$ equal to x''.

We assume that $\mathcal{C}$ communicates with $\mathcal{A}_1$ and $\mathcal{A}_2$ over secure communication channels. This assumption is common in the secure multiparty computation literature. In the context of secure outsourcing, this assumption is also valid, as secure communication is required to protect $\mathcal{C}$'s private sequences against eavesdroppers. The communication between $\mathcal{A}_1$ and $\mathcal{A}_2$ should also be over a secure communication channel. Even though most of the communication between $\mathcal{A}_1$ and $\mathcal{A}_2$ is encrypted using a homomorphic encryption scheme (e.g., see Sect. 3.3), a secure communication channel can prevent replay attacks or person-in-the-middle attacks.

4.1.2 Performance analysis

The local computations done by $\mathcal{C}$ in the above protocol consist of splitting λ and μ and sending the resulting shares to the agents and computing and sending the vectors $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}$. These are done in $O(m+n)$ time and communication.

Each agent mimics mn steps of the dynamic program. During each step, two agents run the secure table lookup protocol once and the minimum finding protocol once. Thus, the communication between $\mathcal{A}_1$ and $\mathcal{A}_2$ for each such step is $O(\sigma^2) + O(1)$. Therefore, the total computation and communication cost for each agent is $O(\sigma^2 mn)$.

From a practical perspective, the multiplicative constant in the "$O(\sigma^2 mn)$" notion is fairly large. Recall that, in Sect. 3.3, $\mathcal{A}_1$ needs $O(\sigma^2)$ homomorphic encryptions and $\mathcal{A}_2$ needs $O(1)$ homomorphic encryptions per execution of the secure table lookup protocol. The minimum finding protocol [2] requires $O(1)$ homomorphic encryptions and $O(1)$ secure comparisons (also known as Yao's millionaire problem [11, 34]; see [15] for the implementation of Yao's millionaire protocol) for each agent. Therefore, the computational load for each agent is at most $O(\sigma^2 mn)$ homomorphic encryptions and $O(mn)$ secure comparisons. The communication cost is the one required for doing the homomorphic encryptions, i.e., it is $O(\sigma^2 mn)$.

4.1.3 An improved version of the protocol

A bottleneck in the above protocol is the split computation of $S(\lambda_i, \mu_j)$: running the secure table lookup protocol at each step of the dynamic program costs an expensive $O(\sigma^2)$. In this subsection, we present a solution that is more efficient by a factor of σ.

Recall that in the dynamic program, M is constructed row by row or column by column. We assume, without loss of generality, that M is computed row by row. We will compute $S(\lambda_i, \mu_j)$ row by row exploiting the fact that all (λ_i, μ_j) in row i have the same λ_i: we will "batch" these table accesses for row i, as we describe next.

Protocol 2. *Batched secure table lookup protocol*

Input $\mathcal{A}_1$ has λ'_i and $\mu' = \mu'_1, \ldots, \mu'_m$, and $\mathcal{A}_2$ has λ''_i and $\mu'' = \mu''_1, \ldots, \mu''_m$, all symbols being over alphabet Σ.

Output $\mathcal{A}_1$ and $\mathcal{A}_2$ each obtain a vector γ' and γ'' of size m such that $\gamma'_j + \gamma''_j = S(\lambda_i, \mu_j)$ for $1 \le j \le m$.

The protocol is:

1. $\mathcal{A}_1$ generates a key pair for a homomorphic semantically secure public-key system and sends the public key to $\mathcal{A}_2$. As before, $E(\cdot)$ denotes encryption with $\mathcal{A}_1$'s public key and $E^{-1}(\cdot)$ decryption with $\mathcal{A}_1$'s private key.

2. $\mathcal{A}_1$ generates a $\sigma \times \sigma$ table $\hat{S}$ with $\hat{S}(k,l)$ equal to $E(S(k+\lambda'_i \bmod \sigma, l))$ for all $0 \le k, l \le \sigma - 1$ and sends that table to $\mathcal{A}_2$.
3. For each $j = 1, \ldots, m$, the next five substeps are carried out to compute the (γ'_j, γ''_j) pair.
   (a) $\mathcal{A}_2$ creates a σ size vector $\mathbf{v}$ equal to the λ''_ith row of the table $\hat{S}$ received in the previous step. Observe that $v_l = E(S(\lambda''_i + \lambda'_i \bmod \sigma, l)) = E(S(\lambda_i, l))$ for $0 \le l \le \sigma - 1$.
   (b) $\mathcal{A}_2$ circularly left-shifts $\mathbf{v}$ by μ''_j positions, so that v_l becomes $E(S(\lambda_i, \mu''_j + l \bmod \sigma))$ for $0 \le l \le \sigma - 1$.
   (c) $\mathcal{A}_2$ generates a random number γ''_j and then updates $\mathbf{v}$ by setting $v_l = v_l * E(-\gamma''_j) = E(S(\lambda_i, \mu''_j + l \bmod \sigma) - \gamma''_j)$ for $0 \le l \le \sigma - 1$. Note that the μ'_jth entry of the resulting $\mathbf{v}$ is now $E(S(\lambda_i, \mu_j) - \gamma''_j)$.
   (d) $\mathcal{A}_1$ uses a 1-out-of-σ oblivious transfer protocol to obtain the μ'_jth entry of $\mathbf{v}$ from $\mathcal{A}_2$ without revealing to $\mathcal{A}_2$ which v_l he received (see, e.g., [28] for many detailed oblivious transfer protocols).
   (e) $\mathcal{A}_1$ decrypts the value he obtained from the oblivious transfer of the previous step and gets $\gamma'_j = S(\lambda_i, \mu_j) - \gamma''_j$. Observe that $\gamma'_j + \gamma''_j = S(\lambda_i, \mu_j)$, as required.

Neither $\mathcal{A}_1$ nor $\mathcal{A}_2$ learned anything about which entry of S was implicitly accessed, or what the value obtained in split fashion was. The communication cost of the above scheme is $O(\sigma^2) + O(\sigma m)$. The size of the alphabet is much smaller than the length of a sequence (e.g., in bioinformatics $\sigma = 4$ whereas a sequence's length is huge). Therefore, the dominant term in the complexity of the above is $O(\sigma m)$.

In the preceding protocol, $\mathcal{A}_1$ computes $O(\sigma^2)$ homomorphic encryptions (for the encryption of the cost table), whereas $\mathcal{A}_2$ computes $O(m)$ homomorphic encryptions (one for each γ''_j) and $O(\sigma m)$ multiplications (of two encrypted items). Note that a homomorphic encryption requires $O(1)$ modular exponentiations. In addition, $\mathcal{A}_1$ and $\mathcal{A}_2$ need to perform a 1-out-of-σ oblivious transfer protocol m times. Naor and Pinkas [20] have constructed an efficient 1-out-of-σ oblivious transfer that requires $\log \sigma$ exponentiations with $O(\sigma)$ communication overhead. Thus, the dominant cost in the computation is $O(m \log \sigma)$ modular exponentiations for each agent. The communication complexity is still $O(m\sigma)$.

The new outsourcing protocol for sequence comparisons is the same as the preliminary protocol in the previous subsection, except for some modifications in the first step of the protocol, titled "mimicking a step of the dynamic program." Recall that the aim of step 1 is to produce a u' with $\mathcal{A}_1$ and a u'' with $\mathcal{A}_2$ such that $u' + u'' = M(i-1, j-1) + S(\lambda_i, \mu_j)$. In the improved protocol, we first run the above batched lookup protocol for row i to produce a γ' for $\mathcal{A}_1$ and a γ'' for $\mathcal{A}_2$, such that $\gamma'_j + \gamma''_j = S(\lambda_i, \mu_j)$ for $1 \le j \le m$. Then, during step 1 of the modified protocol, $\mathcal{A}_1$ sets $u' = M'(i-1, j-1) + \gamma'_j$ and $\mathcal{A}_2$ sets $u'' = M''(i-1, j-1) + \gamma''_j$. Note that, at the end of the new step 1, $u' + u''$ is equal to $M(i-1, j-1) + S(\lambda_i, \mu_j)$, as required. The computational task for the customer in this protocol is the same as in the preliminary version. The computational and communication cost for the agents in this protocol are $\Theta(\sigma mn)$.

In practice, the dominant overhead in the new outsourcing protocol is the $O(mn)$ executions of a 1-out-of-σ oblivious transfer and $O(mn)$ secure comparisons. That is, each agent is required to perform $O(mn \log \sigma)$ modular exponentiations and $O(mn)$ secure comparisons. The communication overhead is $O(\sigma mn)$.

4.2 The case $S(a,b) = |a-b|$

The improvement in this case comes from a more efficient way of computing the split $S(\lambda_i, \mu_j)$ values needed in step 1 of the protocol. Unlike previous sections of the paper, each symbol in λ and μ is split into two numbers that are not modulo σ and can in fact be arbitrary (and possibly negative) integers. The protocol is otherwise the same as in Sect. 4.1.

The main difference is in the first step of subprotocol "mimicking a step of the dynamic program." Note that

$$\begin{aligned} S(\lambda_i, \mu_j) &= |\lambda_i - \mu_j| \\ &= \max(\lambda_i - \mu_j, \mu_j - \lambda_i) \\ &= \max \begin{pmatrix} (\lambda'_i - \mu'_j) + (\lambda''_i - \mu''_j) \\ (\mu'_j - \lambda'_i) + (\mu''_j - \lambda''_i) \end{pmatrix} . \end{aligned}$$

The $S(\lambda_i, \mu_j)$ can be computed as follows: $\mathcal{A}_1$ forms a two-entry vector $\mathbf{v}' = (\lambda'_i - \mu'_j, \mu'_j - \lambda'_i)$, $\mathcal{A}_2$ forms a two-entry vector $\mathbf{v}'' = (\lambda''_i - \mu''_j, \mu''_j - \lambda''_i)$, then $\mathcal{A}_1$ and $\mathcal{A}_2$ use the split maximum finding protocol (described in [2]) to obtain γ' and γ'' such that

$$\gamma' + \gamma'' = \max(\mathbf{v}' + \mathbf{v}'') = |\lambda_i - \mu_j| = S(\lambda_i, \mu_j).$$

Then the first step of the dynamic program can be replaced by $\mathcal{A}_1$ setting $u' = M'(i-1, j-1) + \gamma'$, and $\mathcal{A}_2$ setting $u'' = M''(i-1, j-1) + \gamma''$. As required, $u' + u''$ equals $M(i-1, j-1) + S(\lambda_i, \mu_j)$. As the communication cost of step 1 is now $O(1)$, the total communication cost for the agents is $O(mn)$.

4.3 The case of unit insertion/deletion costs and forbidden substitutions

The improvement in this case directly follows from a technique, given in [2], that we now review. Forbidden substitutions means that $S(a,b)$ is $+\infty$ unless $a = b$ (in which case it is zero because it is a "do nothing" operation). Of course a substitution is useless if its cost is 2 or more (because one might as well achieve the same effect with a deletion followed by an insertion). The protocol is then:

1. For $i = \sigma, \ldots, 1$ in turn, $\mathcal{C}$ replaces every occurrence of symbol i by the symbol $2i$. So the alphabet becomes effectively $\{0, 2, 4, \ldots, 2\sigma - 2\}$.

2. $\mathcal{C}$ runs the protocol given in the previous section for the case of $S(a,b) = |a-b|$, using a unit cost for every insertion and every deletion.

The reason it works is that, after the change of alphabet, $S(a,b)$ is zero if $a = b$ and 2 or more if $a \neq b$, i.e., it is as if $S(a,b) = +\infty$ if $a \neq b$ (recall that a substitution is useless if its cost is 2 or more because one can achieve the same effect with a deletion followed by an insertion).

5 Computing the edit path

We have so far established that the edit distance can be computed in linear space and $O(\sigma mn)$ time and communication. This section deals with extending this to computing, also in split form, the *edit path*, which is a sequence of operations that corresponds to the edit distance (that is, a minimum-cost sequence of operations on λ that turns it into μ). We show that the edit path can be computed by the agents in split form in $O(mn)$ space and in $O(\sigma mn)$ time and communication.

5.1 Review: grid graph view of the problem

The interdependencies among the entries of the M-matrix induce an $(n+1) \times (m+1)$ *grid* directed acyclic graph (grid DAG for short) associated with the string editing problem. It is easy to see that in fact the string editing problem can be viewed as a shortest-path problem on a grid DAG.

Definition 1. *An $l_1 \times l_2$ grid DAG is a directed acyclic graph whose vertices are the $l_1 l_2$ points of an $l_1 \times l_2$ grid and such that the only edges from grid point (i,j) are to grid points $(i, j+1)$, $(i+1, j)$, and $(i+1, j+1)$.*

Figure 1 shows an example of a grid DAG and also illustrates our convention of drawing the points such that point (i,j) is at the ith row from the top and jth column from the left. Note that the top-left point is $(0,0)$ and has no edge entering it (i.e., is a *source*) and that the bottom-right point is (m,n) and has no edge leaving it (i.e., is a *sink*).

We now review the correspondence between edit scripts and grid graphs. We associate an $(n+1) \times (m+1)$ grid DAG G with the string editing problem in the natural way: the $(n+1)(m+1)$ vertices of G are in one-to-one correspondence with the $(n+1)(m+1)$ entries of the M-matrix, and the *cost* of an edge from vertex (k,l) to vertex (i,j) is equal to $I(\mu_j)$ if $k = i$ and $l = j-1$, to $D(\lambda_i)$ if $k = i-1$ and $l = j$, to $S(\lambda_i, \mu_j)$ if $k = i-1$ and $l = j-1$. We can restrict our attention to edit paths that are not wasteful in the sense that they do no obviously inefficient moves such as inserting then deleting the same symbol, changing a symbol into a new symbol that they then delete, etc. More formally, the only edit scripts considered are those that apply at most one edit operation to a given symbol occurrence. Such edit scripts that transform λ into μ or vice versa are in one-to-one correspondence to the weighted paths of G that originate at the source (which corresponds to $M(0,0)$) and end on the sink (which corresponds to $M(n,m)$). Thus, any complexity bounds we establish for the problem of finding a shortest (i.e., least-cost) source-to-sink path in an $(n+1) \times (m+1)$ grid DAG G extends naturally to the string editing problem.

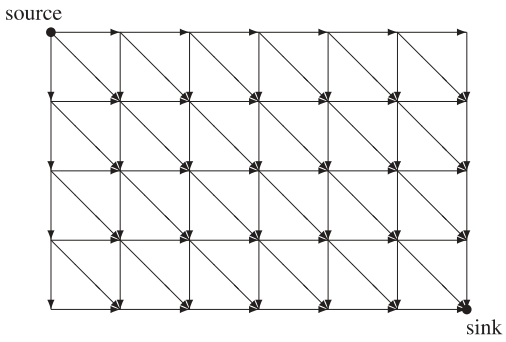

Fig. 1. Example of a 5×7 grid DAG

At first glance it looks like "remembering" (in split form), for every entry $M(i,j)$, which of $\{M(i-1,j-1), M(i-1,j), M(i,j-1)\}$ "gave it its value" would solve the problem of obtaining the source-to-sink shortest path we seek. That is, if we use $P(i,j)$ (where P is mnemonic for "parent") to denote that element $(k,l) \in \{(i-1,j-1),(i-1,j),(i,j-1)\}$ such that the edit path goes from vertex (k,l) to vertex (i,j) in the $(n+1) \times (m+1)$ grid graph that implicitly describes the problem, then all we need to do is store matrix P in split fashion as $P' + P''$. However, this does not work because it would reveal the edit path to both agents: getting that edit path would require starting at vertex (n,m) and repeatedly following the parent until vertex $(0,0)$ is reached, which appears impossible to do without revealing the path to the agents. To get around this difficulty, we use a different approach that we develop next.

5.2 Backward version of the protocol

As mentioned above, the protocol of this section is not an end in itself but will later serve (when used in judicious conjunction with the protocol of the previous section) to efficiently compute the edit path.

The protocol we presented works by computing (in split form) a matrix M such that $M(i,j)$ contains the length of a shortest path from vertex $(0,0)$ to vertex (i,j) in the grid graph G. We call this the *forward protocol* and henceforth denote the M-matrix as M_F, where the subscript F is a mnemonic for "forward." Let G^R denote the *reverse* of G, i.e., the graph obtained from G by reversing the direction of every edge (so the horizontal edges in G^R are pointing leftwards and the vertical edges are pointing upwards – the opposite of the directions shown for G

in Fig. 1). Clearly, in G^R, it is vertex (m,n) that is the source and $(0,0)$ that is the sink, and every v-to-w shortest path in G^R corresponds to a similar shortest path in G *but in the backwards direction* (i.e., w-to-v); this is why we use M_B to denote the matrix that is to G^R what matrix M_F was to graph G (the subscript B is a mnemonic for "backward"). Therefore, $M_B(i,j)$ denotes the length of a shortest path in G^R from the source of G^R (vertex (m,n)) to vertex (i,j), which of course is equal to the length of a shortest path in G from (i,j) to (m,n). The edit distance we seek is therefore $M_B(0,0)$ $(= M_F(n,m))$. Defined in terms of the two input strings, $M_B(i,j)$ is the edit distance from the suffix of λ of length $n-i$ to the suffix of μ of length $m-j$. Therefore, computing M_B in an analogous manner to the computation of M_F now involves filling in its entries by *decreasing* row and column order (rather than increasing, as with M_F). The details, given next, assume that the customer and two agents have already computed M_F in split fashion (so there is no need for a second initialization).

Protocol 3. *Protocol for computing M_B*

1. $\mathcal{A}_1$ sets $M'_B(n,m)=0$, and $\mathcal{A}_2$ sets $M''_B(n,m)=0$. Note that $M_B(n,m)=0$ as expected.
2. For $0\le i\le n-1$, $\mathcal{A}_1$ computes $M'_B(i,m)=M'_F(n,0)-M'_F(i,0)$ and $\mathcal{A}_2$ computes $M''_B(i,m)=M''_F(n,0)-M''_F(i,0)$. Note that $M_B(i,m)=M_F(n,0)-M_F(i,0)=\sum_{k=i+1}^{n} D(\lambda_k)$ for $0\le i\le n-1$.
3. For $0\le j\le m-1$, $\mathcal{A}_1$ computes $M'_B(n,j)=M'_F(0,m)-M'_F(0,j)$ and $\mathcal{A}_2$ computes $M''_B(n,j)=M''_F(0,m)-M''_F(0,j)$. Note that $M_B(n,j)=M_F(0,m)-M_F(0,j)=\sum_{k=j+1}^{m} I(\mu_k)$ for $0\le j\le m-1$.
4. As

$$M_B(i,j)=\min\begin{cases} M_B(i+1,j+1)+S(\lambda_i,\mu_j)\\ M_B(i+1,j)+D(\lambda_i)\\ M_B(i,j+1)+I(\mu_j)\end{cases};$$

if $M_B(i+1,j)$, $M_B(i,j+1)$, and $M_B(i+1,j+1)$ have already been computed, $M_B(i,j)$ can be computed using similar techniques that compute $M_F(i,j)$: $\mathcal{A}_1$ and $\mathcal{A}_2$ compute $S(\lambda_i,\mu_j)$, $D(\lambda_i)$, and $I(\mu_j)$ in split form, and run a minimum finding protocol. M_B's entries can be filled in row by row or column by column in decreasing row and column order.

Note that $M_F(i,j)+M_B(i,j)$ is the length of a shortest source-to-sink path *that is constrained to go through vertex* (i,j) and hence might not be the shortest possible source-to-sink path. However, if the shortest source-to-sink path goes though vertex (i,j), then $M_F(i,j)+M_B(i,j)$ is equal to the length of the shortest path. We use M_C to denote M_F+M_B (where subscript C is mnemonic for "constrained").

The protocol below finds (in split fashion), for each row i of M_C, the column $\theta(i)$ of the minimum entry of that row, with ties broken in favor of the rightmost such entry; note that $M_C(i,\theta(i))$ is the edit distance $M_F(n,m)$. Computing (in split fashion) the θ function is an implicit description of the edit path:

- If $\theta(i+1)=\theta(i)=j$, then the edit path "leaves" row i through the vertical edge from vertex (i,j) to vertex $(i+1,j)$ (the cost of that edge is, of course, the cost of deleting λ_{i+1}).
- If $\theta(i+1)=\theta(i)+\delta$, where $\delta>0$, then the client can "fill in" in $O(\delta)$ time the portion of the edit path from vertex $(i,\theta(i))$ to vertex $(i+1,\theta(i)+\delta)$ (because such a "thin" edit distance problem on a $2\times\delta$ subgrid is trivially solvable in $O(\delta)$ time). The cumulative cost of all such "thin problem solutions" is $O(m)$ because the sum of all such δs is $\le m$.

5.3 Edit path protocol

The steps of the protocol for computing the edit path are:

1. $\mathcal{C}$, $\mathcal{A}_1$, and $\mathcal{A}_2$ conduct the edit distance protocol as described in Sect. 4 to compute M_F in split fashion, i.e., $\mathcal{A}_1$ gets M'_F and $\mathcal{A}_2$ gets M''_F such that $M_F=M'_F+M''_F$.
2. Similarly, $\mathcal{A}_1$ and $\mathcal{A}_2$ conduct the backward version of the edit distance protocol and compute M_B in split fashion. As a result, $\mathcal{A}_1$ gets M'_B and $\mathcal{A}_2$ gets M''_B.
3. $\mathcal{A}_1$ computes $M'_C=M'_F+M'_B$, and $\mathcal{A}_2$ computes $M''_C=M''_F+M''_B$. Note that $M'_C+M''_C$ equals M_C.
4. For $i=0,\ldots,n$ in turn, the following steps are repeated:
   (a) $\mathcal{A}_1$ picks the ith row from M'_C, denoted as $(v'_0,\ldots,v'_m)$, and $\mathcal{A}_2$ picks the ith row from M''_C, denoted as $(v''_0,\ldots,v''_m)$.
   (b) For $0\le j\le m$, $\mathcal{A}_1$ sets $v'_j=(m+1)*v'_j$ and $\mathcal{A}_2$ sets $v''_j=(m+1)*v''_j+(m-j)$; note that $v'_j+v''_j=(m+1)*M_C(i,j)+(m-j)$. Also observe that, if $M_C(i,j)$ is the rightmost minimum entry in row i of M_C, then $v'_j+v''_j$ is now the *only* minimum entry among all $j\in[0..m]$; in effect, we have implicitly broken any tie between multiple minima in row i in favor of the rightmost one (which has the highest j and therefore is "favored" by the addition of $m-j$). Note, however, that breaking the tie through this addition of $m-j$ without the prior scaling by a factor of $m+1$ would have been erroneous, as it would have destroyed the minima information.
   (c) $\mathcal{A}_1$ and $\mathcal{A}_2$ run the minimum finding protocol for split data (described in [2]) on their respective $(v'_0,\ldots,v'_m)$ and $(v''_0,\ldots,v''_m)$. As a result, $\mathcal{A}_2$ gets an x' and $\mathcal{A}_2$ gets and x'' whose sum $x'+x''$ is $\min(v'_0+v''_0,\ldots,v'_m+v''_m)$.
   (d) $\mathcal{A}_1$ and $\mathcal{A}_2$ send x' and (respectively) x'' to $\mathcal{C}$. $\mathcal{C}$ computes

$$\begin{aligned} p_i &= x'+x'' \bmod (m+1)\\ &= ((m+1)M_C(i,\theta(i))+(m-\theta(i))) \bmod (m+1)\\ &= m-\theta(i)\end{aligned}$$

   and therefore obtains $\theta(i)=m-p_i$.

5. As mentioned earlier, given $\theta(0), \ldots, \theta(m)$, $\mathcal{C}$ can compute the edit path in $O(m)$ additional time.

5.3.1 Performance analysis

The computation by the client includes initializing the edit distance protocol (step 1) and computing the edit path from the $\theta(i)$s (step 5). It can be done in $O(m+n)$ time and communication.

The agents run the edit distance protocol twice (steps 1 and 2) and the minimum finding protocol $n+1$ times (step 4). Each edit distance protocol can be done in $O(\sigma mn)$ time and communication, and each minimum finding protocol needs $O(m)$ time and communication. Therefore, the total computation and communication cost for each agent is $O(\sigma mn)$. The space complexity for each agent is $O(mn)$ as the agents need to store M_C in split fashion.

From a practical perspective, each edit distance protocol takes $O(mn)$ executions of a 1-out-of-σ oblivious transfer and $O(mn)$ secure comparisons, and each minimum finding protocol takes $O(m)$ secure comparisons and $O(m)$ homomorphic encryptions. Therefore, the dominant cost for $\mathcal{A}_1$ and $\mathcal{A}_2$ in the edit path protocol is the $O(mn)$ executions of the 1-out-of-σ oblivious transfer, $O(mn)$ executions of secure comparisons, and $O(mn)$ executions of homomorphic encryptions.

5.3.2 An alternative solution with $O(m+n)$ space

The space complexity improvement in this solution comes from the fact that row i of M_C can be computed on the fly; therefore, there is no need for $\mathcal{A}_1$ and $\mathcal{A}_2$ to store M'_C and (respectively) M''_C. The protocol is the same as above, except for the following changes (where we assume, without loss of generality, that $n \leq m$ – in the other case simply change every "row by row" into a "column by column"):

1. Steps 1–3 become "$\mathcal{C}$ splits λ and μ and initializes M_F."
2. In step 4(a), $\mathcal{A}_1$ and $\mathcal{A}_2$ compute M_F row by row from row 1 to row i; similarly $\mathcal{A}_1$ and $\mathcal{A}_2$ compute M_B row by row from row n to row i. Thus $\mathcal{A}_1$ and $\mathcal{A}_2$ have row i of M_C in split fashion.

Note that the total space complexity for each agent is $O(m+n)$, and the total computation and communication cost for each agent is $O(\sigma mn^2)$, as M_F and M_B need to be computed for $n+1$ times. Restating the results for arbitrary m and n: The agents can compute the edit path within a time complexity of $O(\sigma mn \min\{m,n\})$ and in $O(m+n)$ space.

6 Conclusion and future work

We gave efficient protocols that enable a customer to securely outsource sequence comparisons to two remote agents such that the agents learn nothing about the customer's two private sequences or the result of the comparison. The local computations done by the customer are linear in the size of the sequences, and the computational cost and amount of communication done by the external agents are close to the time complexity of the best known algorithm for solving the problem on a single machine. Such protocols hold the promise of allowing weak computational devices to avail themselves of the computational, storage, and bandwidth resources of powerful remote servers without having to reveal to those servers their private data or the outcome of the computation.

Future work includes developing efficient outsourcing protocols for other computation-intensive problems, such as image processing problems, other biological computational problems, etc. Our current secure outsourcing model requires two noncolluding agents. Future work also includes designing outsourcing protocols using a single agent or using n agents such that the protocols are secure against collusion of up to t agents.

Acknowledgements. We would like to thank the anonymous reviewers for their helpful comments.

References

1. Aho AV, Hirschberg DS, Ullman JD (1976) Bounds on the complexity of the longest common subsequence problem. J ACM 23(1):1–12
2. Atallah MJ, Kerschbaum F, Du W (2003) Secure and private sequence comparisons. In: 2nd ACM workshop on privacy in electronic society
3. Atallah MJ, Li J (2004) Secure outsourcing of sequence comparisons. In: 4th workshop on privacy enhancing technologies
4. Atallah MJ, Pantazopoulos KN, Rice J, Spafford EH (2001) Secure outsourcing of scientific computations. Adv Comput 54(6):215–272
5. Beguin P, Quisquater JJ (1995) Fast server-aided RSA signatures secure against active attacks. In: Advances in Cryptology – Crypto 1995. Lecture notes in computer science, vol 963. Springer, Berlin Heidelberg New York, pp 57–69
6. Boneh D, Crescenzo GD, Ostrovsky R, Persiano P (2004) Public-key encryption with keyword search. In: Advances in Cryptology – Eurocrypt 2004. Lecture notes in computer science, vol 3027. Springer, Berlin Heidelberg New York, pp 506–522
7. Cachin C (1999) Efficient private bidding and auctions with an oblivious third party. In: 6th ACM conference on computer and communications security, pp 120–127
8. Du W, Atallah MJ (2000) Protocols for secure remote database access with approximate matching. In: 1st ACM workshop on security and privacy in e-commerce
9. Fischlin M (2001) A cost-effective pay-per-multiplication comparison method for millionaires. In: RSA Security 2001 Cryptographer's Track. Lecture notes in computer science, vol 2020. Springer, Berlin Heidelberg New York, pp 457–471
10. Foster I, Kesselman C (ed) (1999) The grid: blueprint for a new computing infrastructure. Morgan Kaufmann, San Francisco
11. Goldreich O (2004) Foundations of cryptography. Basic applications, vol 2. Cambridge University Press, Cambridge, UK
12. Kawamura SI, Shimbo A (1993) Fast server-aided secret computation protocols for modular exponentiation. IEEE J Select Areas Commun 11(5):778–784
13. Landau G, Vishkin U (1986) Introducing efficient parallelism into approximate string matching and a new serial algorithm. In: 18th ACM STOC, pp 220–230

14. Lim CH, Lee PL (1995) Security and performance of server-aided RSA computation protocols. In: Advances in Cryptology – Crypto 1995. Lecture notes in computer science, vol 963. Springer, Berlin Heidelberg New York, pp 70–83
15. Malkhi D, Nisan N, Pinkas B, Sella Y (2004) Fairplay – a secure two-party computation system. In: Usenix Security '2004, pp 287–302
16. Martinez HM (ed) (1984) Mathematical and computational problems in the analysis of molecular sequences. Bull Math Biol 46(4) [Special Issue Honoring M.O. Dayhoff]
17. Masek WJ, Paterson MS (1980) A faster algorithm computing string edit distances. J Comput Syst Sci 20:18–31
18. Matsumoto T, Kato K, Imai H(1988) Speeding up secret computations with insecure auxiliary devices. In: Advances in Cryptology – Crypto 1988. Lecture notes in computer science, vol 403. Springer, Berlin Heidelberg New York, pp 497–506
19. Naccache D, Stern J (1998) A new cryptosystem based on higher residues. In: 5th ACM conference on computer and communications security, pp 59–66
20. Naor M, Pinkas B (1999) Oblivious transfer and polynomial evaluation. In: 31st symposium on theory of computer science, pp 245–254
21. Needleman SB, Wunsch CD (1973) A general method applicable to the search for similarities in the amino-acid sequence of two proteins. J Mol Biol 48:443–453
22. Ogata W, Kurosawa K (2004) Oblivious keyword search. J Complex 20:356–371
23. Okamoto T, Uchiyama S (1998) A new public-key cryptosystem as secure as factoring. In: Advances in Cryptology – Eurocrypt 1998. Lecture notes in computer science, vol 1403. Springer, Berlin Heidelberg New York, pp 308–318
24. Pfitzmann B, Waidner M (1992) Attacks on protocols for server-aided RSA computations. In: Advances in Cryptology – Eurocrypt 1992. Lecture notes in computer science, vol 658. Springer, Berlin Heidelberg New York, pp 153–162
25. Rivest RL, Adleman L, Dertouzos ML(1978) On data banks and privacy homomorphisms. In: DeMillo R (ed) Foundations of secure computation. Academic, New York, pp 169–177
26. Sankoff D (1972) Matching sequences under deletion-insertion constraints. Proc Natl Acad Sci USA 69:4–6
27. Sankoff D, Kruskal JB (ed) (1983) Time warps, string edits and macromolecules: the theory and practice of sequence comparison. Addison-Wesley, Reading, MA
28. Schneier B (1995) Applied cryptography: protocols, algorithms, and source code in C, 2nd edn. Wiley, New York
29. Sellers PH (1974) An algorithm for the distance between two finite sequences. J Combinator Theory 16:253–258
30. Sellers PH(1980) The theory and computation of evolutionary distance: pattern recognition. J Algorithms 1:359–373
31. Ukkonen E (1985) Finding approximate patterns in strings. J Algorithms 6:132–137
32. Wagner RA, Fischer MJ(1974) The string to string correction problem. J ACM 21(1):168–173
33. Wong CK, Chandra AK(1976) Bounds for the string editing problem. J ACM 23(1):13–16
34. Yao A (1982) Protocols for secure computations. In: 23th IEEE symposium on foundations of computer science, pp 160–164