Communication-Efficient Private Protocols for Longest Common Subsequence *

Matthew Franklin, Mark Gondree, and Payman Mohassel

Department of Computer Science
University of California, Davis
{franklin, gondree, mohassel}@cs.ucdavis.edu

Abstract. We design communication efficient two-party and multi-party protocols for the longest common subsequence (LCS) and related problems. Our protocols achieve privacy with respect to passive adversaries, under reasonable cryptographic assumptions. We benefit from the somewhat surprising interplay of an efficient block-retrieval PIR (Gentry-Ramzan, ICALP 2005) with the classic "four Russians" algorithmic design. This result is the first improvement to the communication complexity for this application over generic results (such as Yao's garbled circuit protocol) and, as such, is interesting as a contribution to the theory of communication efficiency for secure two-party and multiparty applications.

1 Introduction

We design communication efficient two-party and multi-party protocols for two variants of the longest common subsequence (LCS) problem, and related problems. The first variant returns only the length of the LCS, while the second outputs the string encoding the subsequence itself. Previous work on this topic [25, 34, 10] has focused on implementing basic dynamic programming algorithms privately, using techniques that each achieve $O(n^2)$ communication complexity (where n is the length of each input string). Jha, Kruger, and Shmatikov [25] demonstrate that some of these solutions may be practically quite efficient. This previous work, however, does not improve on the asymptotic communication complexity of generic solutions such as Yao's garbled circuit protocol [36]. Thus, our work is both theoretically and practically interesting since it can be interpreted as a new upper bound on the communication complexity of this problem.

Traditionally, the method known as the "four Russians" technique yields only a logarithmic improvement in the running time of the dynamic programming solution for LCS. Somewhat surprisingly, we show

* This is the full version of an article [18] to appear in *CT-RSA 2009*.

how to take advantage of an efficient block-retrieval PIR due to Gentry and Ramzan [21] and use the "four Russians" technique to obtain a communication efficient private protocol for the LCS problem. Specifically, we design protocols that achieve $O(n^2/t)$ total communication cost, for any positive $t < n$.

In our protocol, the computational cost of one party increases (anywhere between linearly and exponentially, depending on the choice of parameter t), while the remaining parties' computation costs reduce to $O(n^2/t)$. In the most practical setting, where each party performs only a polynomial amount of computation, we achieve a new, sub-quadratic upper bound for the private protocol's communication complexity.

If some participant can perform more work, our protocol becomes even more communication efficient. This setting, where an asymmetric work load yields communication savings for all participants, may be quite advantageous. This client-server setting may even be realistic, given the large volume of genomic data held by central entities. We also show how to outsource the work needed by our protocol to a set of powerful, dedicated yet untrusted servers.

Motivation: Performing different computational tasks on large biological databases is becoming a more common practice in both public and private institutions. The FBI maintains a database of over four million DNA profiles of criminal offenders, crime scene evidence, and missing persons in its CODIS system [2], and uses the data for forensic studies and DNA-based identification. deCODE Genetics [3], a biopharmaceutical company which studies genomic data for drug discovery and development, has collected the genotypic and medical data of over 50 percent of the population in Iceland. Similar endeavors seek to make these types of databases available for scientific study [5].

The genomic data stored in these databases may be extremely sensitive: an individual's DNA sequence reveals a great deal of information regarding that individual's health, background, and physical appearance [1, 4]. It has been shown that a sequence can be linked to the corresponding individual simply by recognizing the presence of certain markers [27]. Protecting a patient's privacy when working with genomic data is recognized as a major challenge for the biomedical research community [8, 35]. Furthermore, in the United States, HIPAA's Privacy Rule [32] mandates that a patient's identity must be protected when their data (including genomic data) is shared; failure to assure this may result in legal action, fines, revocation of government funding, and imprisonment.

The pioneering work of Jha, Kruger, and Shamtikov [25] and others [34, 10] has recognized the need for private and efficient computation on genomic data in general, and the LCS and edit distance problems in particular. We hope our results continue to motivate improvements to the state-of-the-art in this domain.

2 Related Work

It is possible to use generic solutions or pre-existing protocols to solve the problems considered in this paper. However, we desire protocols that are efficient in terms of communication complexity (the total number of bits sent and received by the participants in the protocol).

While the theoretical lower bound[1] for the running time of an algorithm that solves the LCS problem for two strings of length n is $\Omega(|\Sigma|n)$ for a fixed alphabet Σ [6], the theoretically fastest *known* algorithm solving this problem is that of Masek and Patterson [28], achieving $O(n^2/\log n)$. The circuit simulating this algorithm, however, is of size $O(n^2)$. Thus, generic compilers like Yao's garbled circuit protocol [36] would yield communication complexity $O(n^2)$. Naor and Nissim's communication preserving compiler [30] may yield a protocol with $O(n)$ communication complexity, but the protocol would require on-line work that is exponential in n.

We can reduce the LCS problem to the shortest path problem, and then use secure matrix multiplication protocols, *e.g.* [26], or secure graph protocols [11] to recover a solution. The reduction, however, increases the input size of the problem, so that instead of an input of size $\Theta(n)$, we now must consider a graph or matrix of size $\Theta(n^2)$. Even assuming we have access to private protocols for matrix multiplication or shortest path with optimal communication efficiency, using this reduction will yield a protocol with $\Omega(n^2)$ communication complexity.

We could also re-use secure protocols implementing the Needleman-Wunsch algorithm (or a variation of it, the Smith-Waterman algorithm) [10, 34, 25]. These algorithms solve a generalization of the edit distance problem, where insertions and deletions have variable costs. When all costs are 1, these algorithms can be used to directly solve the problems we consider here. Szajda, Pohl, Owen, and Lawson [34] provide a heuristic protocol for Smith-Waterman attaining heuristic security, which fails to meet the correctness or privacy needs considered here. Atallah, Kerschbaum, and Du [10] and Jha, Kruger, and Shmatikov [25] provide protocols whose

[1] This lower bound becomes $\Omega(n^2)$ when the alphabet is not fixed, and the basic operations considered are comparisons.

communication complexities are $O(n^2)$, meeting but not improving upon the asymptotic efficiency of the generic solution, using Yao's garbled circuit protocol.

3 Notations and Definitions

Notation $\Omega(f)$ denotes that the asymptotic lower bound f is tight; $\Theta(f)$, means that f is both a lower bound and an upperbound, and $\widetilde{O}(f)$ denotes the asymptotic upper bound $O(f)$, ignoring polylog(f) factors.

Sharing Values We take advantage of two simple sharing schemes, XOR sharing and additive sharing, in our protocols. Alice and Bob XOR share a value c, if Alice holds the value a and Bob holds the value b, such that $a \oplus b = c$. Similarly, an integer c is additively shared between the parties if $a + b \bmod N = c$, where N is a properly chosen and publicly known integer.

Security We prove our protocols secure against a passive adversary (also referred to as semi-honest) who follows the steps of the protocol but tries to learn additional information based on the messages he receives throughout the protocol. The security in this model is defined by requiring that any adversary in the real protocol, can be simulated by an adversary in an ideal world where parties send their inputs to a trusted party who computes and sends back their corresponding outputs. For a more formal definition, we refer the reader to [22, Volume 2]. Central to our security claims is the following composition theorem.

Theorem 1 (Composition for Passive Adversaries [22]). *Suppose that g is privately reducible to f and that there exists a protocol for privately computing f. Then, there exists a protocol for privately computing g.*

Using the above theorem, along with simple hybrid arguments, it is straight-forward to prove our protocols private against a passive adversary as long as our subprotocols are private.

4 Longest Common Subsequence

Let A and B be two strings over a fixed alphabet Σ of size σ, with lengths $m = |A|$ and $n = |B|$ (without loss of generality, let $m \leq n$). A subsequence of A is a string X such that A can be transformed into X

by deleting characters from A. A longest common subsequence (LCS) of A and B is a subsequence of both A and B such that no other common subsequence has greater length.

Algorithms that solve the longest common subsequence problem return one or more of the following outputs:

1. The length of the LCS of A and B.
2. A string which is a LCS of A and B.
3. An embedding α, β of a LCS of A and B

Where an embedding $\alpha \in \{0,1\}^m$ and $\beta \in \{0,1\}^n$ are bit-strings which select an LCS from A and B, respectively.

4.1 LCS algorithm using standard dynamic programming techniques

The following dynamic programming algorithm solving the longest common subsequence problem was independently discovered by many researchers, in both computer science and biology. For a standard presentation of this type of dynamic programming solution, see [14, §16.3] or [23, §11.3]. Let L be the $(m+1) \times (n+1)$ matrix whose entries can be computed (row-by-row or column-by-column) using the following:

$$L[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ L[i-1, j-1] + 1 & \text{if } A[i] = B[j] \\ \max(L[i-1,j], L[i,j-1]) & \text{otherwise} \end{cases}$$

Entry $L[m,n]$ holds the length of the LCS for A and B, and simple deterministic backtracking algorithms exist for recovering the value and/or the embedding of an LCS for A and B.

4.2 LCS algorithm using the "four Russians" technique

Masek and Patterson [28] give a variant on this dynamic programming solution for the LCS problem, using ideas (colloquially known as the "four Russians" technique) introduced by Arlazarov, Dinic, Kronod and Faradzev [9] for boolean matrix multiplication.

Let t be a positive integer. Then, the boundaries of the $t \times t$ block of the dynamic programming table L starting at position (i,j) can be denoted with the following variables (see Figure 1).

$$N(i,j) = (L[i,j], L[i,j+1], \ldots, L[i,j+t])$$

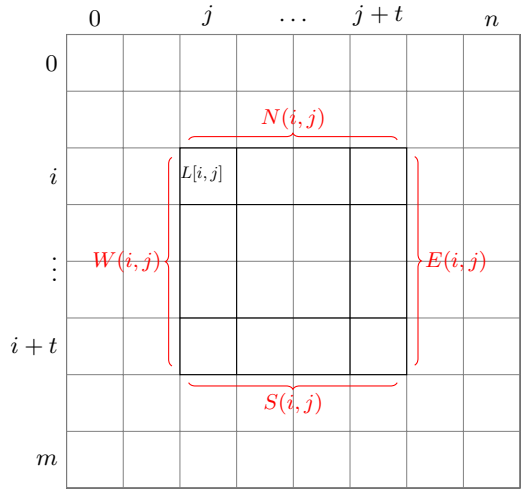

Fig. 1. The $t \times t$ block beginning at position (i, j) in the $(m + 1) \times (n + 1)$ matrix L.

$$W(i,j) = (L[i,j], L[i+1,j], \ldots, L[i+t,j])$$
$$S(i,j) = (L[i+t,j], L[i+t,j+1], \ldots, L[i+t,j+t])$$
$$E(i,j) = (L[i,j+t], L[i+1,j+t], \ldots, L[i+t,j+t])$$

Let the *offset vectors* $I_1(i,j), I_2(i,j) \in \{0,1\}^t$ be defined as

$$I_1(i,j)[k] = \begin{cases} 0 & \text{for } k = 1 \\ N(i,j)[k] - N(i,j)[k-1] & \text{for } 1 < k \leq t \end{cases}$$

$$I_2(i,j)[k] = \begin{cases} 0 & \text{for } k = 1 \\ W(i,j)[k] - W(i,j)[k-1] & \text{for } 1 < k \leq t \end{cases}$$

It is simple to check that consecutive values in the L matrix increase by at most one, so the offsets are indeed bit-vectors.

The basic observation underlying the four Russians technique is that the values $E(i,j)$ and $S(i,j)$ are completely determined by $A[i,\ldots,i+t]$, $B[j,\ldots,j+t]$, $I_1(i,j)$, $I_2(i,j)$, and $L[i,j]$. Denote this basic block functionality as $\text{bbf}(A[i,\ldots,i+t], B[j,\ldots,j+t], I_1(i,j), I_2(i,j), L[i,j]) = (E(i,j), S(i,j))$. Thus, we can compute the entire dynamic programming table in the following manner:

1. **Pre-processing**: pre-compute all possible $t \times t$ blocks, by considering all possible t-length strings and offset vectors, but assume the

first value of the block is 0. That is, generate a table summarizing $\mathrm{bbf}(\cdot,\cdot,\cdot,\cdot,0)$. Note that $\mathrm{bbf}(\cdot,\cdot,\cdot,\cdot,0)+C = \mathrm{bbf}(\cdot,\cdot,\cdot,\cdot,C)$; *i.e.*, if the first value of the block is C and not zero, the pre-computed outputs differ from the desired outputs in every place by the additive term C. The number of entries in this table is $\sigma^{2t}2^{2t}$.

2. **Rebuilding the L matrix**: Consider the L matrix to be composed of $t \times t$ blocks that overlap in one row and one column with each other. Retrieve $\mathrm{bbf}(A[i,\ldots,i+t], B[j,\ldots,j+t], I_1(i,j), I_2(i,j), 0) = (\widetilde{E}(i,j), \widetilde{S}(i,j))$ by looking up the appropriate pre-computed block. Any $L[i',j']$ entry situated on the boundary of this block can be calculated by adding $L[i,j]$ to the appropriate value from the retrieved $\widetilde{E}(i,j)$ or $\widetilde{S}(i,j)$ vectors. Iterate in this fashion, using $\widetilde{S}(i,j), \widetilde{E}(i,j)$ to determine $I_1(i+t,j), I_2(i,j+t)$, each time considering the first value of the block to be 0 during look-up and then compensating by adding back the appropriate additive term.

Essentially, we have reduced our $m \times n$ matrix to an $m/t \times n/t$ matrix. In the unit-cost RAM model, partially filling out the L matrix in this fashion takes $O(mn/t^2)$ time.

5 Communication Efficient Protocols for Private LCS

Here, we build a private protocol for determining the length of the LCS. Later, we provide a deterministic backtracking algorithm for privately recovering an actual LCS or LCS embedding.

Definition 1 (Private LCS-length). *A protocol is a private LCS-length protocol between two parties (one holding a private input string A, the other holding a private input string B) if the protocol outputs the length of the LCS of A and B, but reveals no information to a passive adversary other than what she can learn from the output.*

Below we provide details describing a private LCS-length protocol. At a high-level, the protocol executes the algorithm of Masek and Patterson described earlier, but each party holds shares of the L matrix. Basic block function table look-ups are performed using a communication efficient private block retrieval scheme, while the remaining computations are designed to be performed by the parties locally.

Private block retrieval (PBR) was first introduced in the original PIR paper of Chor *et al.* [13]. A PBR scheme is essentially a private information retrieval (PIR) scheme that allows the chooser to retrieve ℓ-bit

database entries, as opposed to bit entries. We call such a scheme a symmetric PBR (SPBR) scheme if it also provides privacy for the server (database).

Any secure two-party PBR schemes can be transformed into an SPBR scheme that provides this functionality, via the Naor-Pinkas transform [31], the Aiello-Ishai-Reingold transform [7], or zero-knowledge proofs. We note that, excluding zero-knowledge based techniques, these transforms incur no loss of efficiency for the SPBR.

1. **Pre-computation**: Alice pre-computes the table summarizing the function $\mathrm{bbf}(\cdot,\cdot,\cdot,\cdot,0)$, in the following way. Let the binary representation of the inputs $A[i,\ldots,i+t]$, $B[j,\ldots,j+t]$, $I_1(i,j)$, $I_2(i,j)$ be x_1, x_2, x_3, x_4. The resulting index $x_1||x_2||x_3||x_4$ is a bit-string of length $2\log(\sigma)t + 2t$. At this index, Alice stores the following:
   - $\widetilde{E}(i,j), \widetilde{S}(i,j)$.
   - The offset vectors $I_1(i{+}t,j), I_2(i,j{+}t)$ associated with $\widetilde{E}(i,j), \widetilde{S}(i,j)$.

   Clearly these two vector pairs are redundant because given either pair we can compute the other, but as each is shared using a different sharing scheme, holding both will make local computations easier, later. Because the first entry of the pre-computed block is set to zero, the values of $\widetilde{E}(i,j), \widetilde{S}(i,j)$ will always be in the interval $[0,t]$. This means each entry of our look-up table will be a bit-string of length $2t + 2t\log t$.
2. **Accessing the basic block function**: From previous database accesses, Alice and Bob will hold XOR shares of $I_1(i+t,j) = x_3$ and $I_2(i{+}t,j) = x_4$. In all cases, x_1 is known only to Alice and x_2 is known only to Bob. Thus, Alice can simply consider her share of x_2 to be $0^{t\log\sigma}$, and similarly for Bob. This allows Alice and Bob to locally concatenate their values, producing valid XOR shares of the next index $x_1||x_2||x_3||x_4$. The database defining the basic block function is accessed each time in the following way:
   (a) Alice picks some random value and locally blinds each of her database's entries by this value. Particularly, XORing by a random string is used for each offset vector, and adding a random value is used for each $\widetilde{E}(i,j)$ and $\widetilde{S}(i,j)$.
   (b) Alice locally permutes her database by her share of $x_1||x_2||x_3||x_4$.
   (c) Alice and Bob engage in a SPBR protocol using Bob's index share as input, allowing Bob to recover the blinded entries. Alice's shares are the random values (strings) she added (XORed) in the blinding step. Thus, Alice and Bob hold valid shares of the entries of interest.

3. **Reconstructing** L: As in the original Masek and Patterson algorithm, after the appropriate entries of the basic block function are retrieved, Alice and Bob can (non-interactively) use the additive shares of the values to compute shares of entry $L[m, n]$. They then privately exchange these shares to recover $L[m, n]$, the length of the LCS of A and B.

Theorem 2. *The above protocol is a private LCS-length protocol for inputs of length n and m, assuming we have a secure SPBR scheme for a database of N entries each of size ℓ.*

Proof. (sketch) The above security claim follows from the security of the share conversion protocols, our black-box use of SPBR, and from general composition theorems [12, 22]. How to specify the parameters N and ℓ to attain the efficiency we desire is explained in the complexity analysis, below.

5.1 Analysis

Theorem 3. *Assume we have an SBR scheme such that: (1) for a database of size N, on query i the scheme returns the i-th ℓ-bit entry of the database for integer parameter ℓ with $2^\ell < N$, and (2) the scheme has $g(N, \ell) = O(\log N)$ communication complexity and requires $f(N, \ell) = O(N)$ work.*

Then the described protocol for LCS-length has the following complexities, as a function of the parameter t.

Communication Complexity	*Round Complexity*	*Alice's Work*	*Bob's Work*
$O(mn/t)$	$O(m/t + n/t)$	$\widetilde{O}(t2^t)$	$O(mn/t)$

Proof. The above protocol fills out the L matrix as in the Masek-Patterson algorithm after mn/t^2 invocations of the SPBR protocol. Naively, the protocol proceeds (row-by-row or column-by-column) in mn/t^2 rounds. In a straight-forward manner, we can parallelize many of these steps and improve this to $m/t + n/t$ rounds.

When we set $N = \ell 2^t$, then the query returns the appropriate ℓ-bit entry of the bbf look-up table of size 2^t. When $\ell = t \log t$, the computational complexity of the protocol is $O(2^{2t} * f(\ell N, \ell) * (2t + 2t \log t)/\ell) = \widetilde{O}(t2^t)$. The total communication complexity is $O(mn/t^2 * g(\ell N, \ell) * (2t + 2t \log t)/\ell) = O(mn/t)$.

Claim. Concretely, the assumptions of Theorems 2 and 3 are satisfied by the SPBR scheme attained by transforming the Gentry-Ramzan PBR scheme [21] to an SPBR scheme using either the Aiello-Ishai-Reingold or Naor-Pinkas transforms. The security of this SPBR scheme is based on the hardness of the "extended decision subgroup problem" and the assumptions required for the transform (*e.g.*, secure homomorphic encryption for Aiello-Ishai-Reingold).

The proposed protocol establishes a useful framework for many dynamic programming algorithms, providing a range of efficiencies based on trading computational complexity for communication complexity. At one extreme, when $t = 1$, the communication and computational complexities resemble that of Yao's garbled circuit protocol. At the other extreme, when $t = n$, the costs match those of the communication preserving compiler of Naor and Nissim [30], achieving $O(m)$ communication complexity (matching the simple theoretic lower bound) but requiring work exponential in n. Between these extremes, the protocol yields a smooth trade-off between work and communication, achieving any sub-quadratic communication complexity at the expense of computational complexity.

6 Private Backtracking in Our Protocol

We have provided a protocol for determining the length of an LCS of A and B privately, although a natural remaining question is to ask is if we can recover the value of an LCS string itself (or the bit-string which encodes how an LCS is embedded in A and B) privately. We call this a private LCS-backtracking protocol.

Definition 2 (Private LCS-backtracking). *A private LCS-backtracking protocol between two parties is one in which (1) the parties hold private input strings A and B, (2) the protocol outputs an LCS (or, embedding of the LCS) for A and B, (3) there is some deterministic algorithm that agrees with the protocol's output for any valid input, (4) the protocol reveals no information to a passive adversary other than what she can learn from the output.*

In fact, with some careful planning, we can modify our LCS-length protocol to build a LCS-backtracking protocol. Our LCS-backtracking protocol relies on no stronger assumptions than our LCS-length protocol, and its communication complexity and work are asymptotically no greater than that of our LCS-length protocol. The full details of this protocol are provided in Appendix A.

7 Related Applications

Our techniques can be viewed as a specialization of Naor-Nissim circuits with look-up tables [30], for a class of dynamic programming problems that have the property that their subproblems can be "efficiently encoded." By this, we mean that the problem can be decomposed into overlapping subproblems of size t, each of which can be encoded using $O(t)$ bits. For the LCS problem, it was essential that adjacent entries in L differed by at most 1, so each $t \times t$ block could be encoded with $O(t)$ bits. If, however, adjacent entries differed by, say, an arbitrary value in $[0, n]$, then any encoding of the subproblem would require $O(t \log n)$ bits; the size of the subproblem's look-up table would be strictly greater than $O(2^t)$, and the cost of iteratively accessing this table would result in no savings when compared to Yao's garbled circuit protocol.

There are natural applications that fall into this general framework of dynamic programming problems with "efficiently encoded" subproblems. We list some below, with remarks on how to solve each with our private protocols for LCS-length and LCS-backtracking, either immediately or with minor modifications.

7.1 Edit Distance

The private LCS-length protocol automatically yields a private algorithm recovering the edit distance (Levenshtein distance), because

$$\text{edit-distance}(A, B) = m + n - 2 * \text{LCS-length}(A, B)$$

This relationship holds because transforming A into B via a series of deletions and insertions is equivalent to deleting $n -$ LCS-length characters, and inserting $m -$ LCS-length characters.

Damerau-Levenshtein distance (where the allowable operations are insertions, deletions, substitutions, or transpositions) is not automatically implied by LCS-length, but it is clear that a small modification of our protocol will suffice. In particular, when pre-computing the $t \times t$ block at position (i, j) in L, in order to consider transposition operations we may need to refer back to data in row $i - 1$ and column $j - 1$. That is, the input to the basic block function will require $4t$ offset vectors (instead of $2t$ vectors), and the $t \times t$ blocks will tile the L matrix overlapping with a previous block in two rows or columns (instead of just one). Such a modification does not change the asymptotic complexity of the resultant protocol.

Ulam's metric is a type of similarity measure or distance metric for permutations, similar to Kendall's τ-metric or Spearman's rank correlation metric. For permutations π, π' of order n, the Levenshtein distance of π and π' is equivalent to the Ulam distance. Thus, our protocol gives two parties the ability to privately compute the Ulam similarity of their ordered preference lists of common elements. Note that this is somewhat different from the setting of Freedman *et al.* [20], which performed a privacy-preserving set intersection to measure the similarity between unordered preference lists of different elements.

7.2 Shortest Common Supersequence

The private LCS-length protocol automatically yields a private algorithm recovering the length of the shortest common supersequence (SCS), because

$$\text{SCS-length}(A, B) = m + n - \text{LCS-length}(A, B)$$

This relationship holds because the SCS is the shortest string containing A and B as subsequences, which can always be obtained by adding to an LCS the extra characters from A and B. However, when the number of strings is greater than 2, there is no longer a relationship between the problems [19].

7.3 Other dynamic programming applications

The longest common substring (LCSS) problem and longest increasing subsequence (LIS) problem both have dynamic programming solutions with similar structure. Our protocol can be used directly or with slight modifications to solve these problems. For example, given the string A and an ordering on Σ, $\sigma_1 < \sigma_2 < \ldots < \sigma_{|\Sigma|}$, we have $\text{LIS}(A) = \text{LCS}(A, B)$ where $B = \sigma_1^{|A|}\sigma_2^{|A|}\cdots\sigma_{|\Sigma|}^{|A|}$. The LCSS problem can be solved by modifying the basic block function to utilize the recurrence

$$\text{LCSS}(i, j) = \begin{cases} \text{LCSS}(i-1, j-1) + 1 & \text{if } A[i] = B[j] \\ 0 & \text{otherwise} \end{cases}$$

Both the LIS and LCSS problems, however, have more efficient non-dynamic programming solutions: LIS can be solved in $O(n \log n)$ time with arrays and binary search; LCSS can be solved in $O(n + m)$ time with suffix trees. While the circuits implementing these solutions are of size $O(mn)$, it is likely that there are more efficient specialized two-party protocols implementing these solutions.

7.4 Private `diff`

The Unix command "`diff` A B" traditionally returns an annotated file showing the least-costly merge of file A and file B. From the annotations, both A and B can be recovered. Thus, a private implementation of `diff` has little utility, as the output leaks both inputs. It may be natural, however, to consider an asymmetric version of `diff`, where Alice does not learn an annotated file merging A and B but instead learns which parts of A have been removed, which parts have been preserved, and where insertions have been made (analogous to an embedding). Modifying LCS-backtracking to achieve this functionality is straight-forward. Similarly, it may be natural to consider calculating statistics related to the `diff` of two files, and not output the merged file itself. For example, the command

$$\texttt{diff}\ A\ B\ \texttt{|}\ \texttt{wc -l}$$

returns the edit distance of files A and B and, as we have seen, is easily calculated using our LCS-length protocol.

A private `diff` protocol may be useful in the situation where Alice and Bob want to collaborate to determine if either one has "plagiarized" the other, without leaking their own proprietary data to the other participant who, necessarily, they suspect may be possible of plagiarism. A realistic scenario similar to this one is discovering GPL violations [33]; the protocol may be a useful discovery mechanism, allowing Bob to inspect if Alice has violated the use restrictions of his software licensed under the GPL (by including it in her proprietary code without redistributing it under the GPL) while at the same time respecting the Alice's potentially non-infringing, proprietary code.

7.5 Multiparty variants of LCS-length and LCS-backtracing

Instead of running a protocol between two parties, it may be desirable to outsource the work to some fixed number of parties m, such that no passive adversary corrupting less than m participants can learn any information beyond the output, for privacy reasons. Alternatively, perhaps neither Alice nor Bob want to play the role of the server (who pre-computes the basic block function table) but instead want to leverage the resources of one or more external servers, without having to trust them. We can generalize our LCS-length and LCS-backtracking protocols to this scenario using a multiparty generalization of indirect indexing [17, 24]. In particular, the multiparty indirect indexing scheme of Ishai *et al.* [24]

is a general construction using any 2-round OT protocol. To achieve a construction with polylogarithmic communication complexity, Ishai *et al.* use an OT scheme based on PIR with length-flexible messages (achieved using a length-flexible homomorphic encryption scheme such as Damgård-Jurik [15, 16]). To achieve multiparty indirect indexing with a strictly logarithmic communication complexity, we can instead use Gentry-Ramzan. With a database of size N, the m-iterated use of Gentry-Ramzan generates messages of length $O(\log Nc^m) = O(\log N)$ for a small constant $c < 8$ [29]. Thus, we can use the construction of Ishai *et al.*, instantiated with an appropriate version of Gentry-Ramzan, to perform multiparty indirect indexing, and generalize our protocols to the multiparty scenario with no asymptotic increase to the costs.

7.6 Fixed m-string LCS

In the multiparty version of LCS-length and LCS-backtracking protocols, it is natural to consider the situation were each party i holds a string A_i as her private input, and the m parties wish to compute the length (or embedding or string) of the LCS of the strings $A_1, \ldots, A_m$. The following is a straight-forward generalization of the standard dynamic programming solution for this problem, using an m-dimensional L matrix.

1: **for** $i_1 \leftarrow 1 \ldots |A_1|$, $i_2 \leftarrow 1 \ldots |A_2|$, ..., $i_m \leftarrow 1 \ldots |A_m|$ **do**
2: **if** $(A_1[i_1] = A_2[i_2] = \cdots = A_m[i_m])$ **then**
3: $L[i_1, i_2, \ldots, i_m] \leftarrow L[i_1 - 1, i_2 - 1, \ldots, i_m - 1] + 1$
4: **else**
5: $L[i_1, i_2, \ldots, i_m] \leftarrow \max\{L[i_1 - b_1, i_2 - b_2, \ldots, i_m - b_m] : b_1 b_2 \ldots b_m \in \{0,1\}^m$ is a bit-string of Hamming weight $m - 1\}$
6: **end if**
7: **end for**

As before, adjacent rows and columns never differ by more than 1, so the offset vectors can be efficiently encoded using bit-strings. Thus, the four Russians technique can still be used. We can generalize our protocols to simulate this algorithm in a similarly straight-forward fashion.

8 Conclusion

We have presented a private protocol for the longest common subsequence problem, a classic problem for both computer scientists and biologists.

While the "four Russians" technique traditionally offers only a logarithmic savings in work for the traditional dynamic programming algorithm for LCS, we use it to achieve a smooth trade-off between communication complexity and work, filling in the theoretical "terrain" between polynomial-work compilers (like Yao) and exponential-work compilers (like Naor-Nissim). Our protocol features an asymmetry in the work required from each participant, making it appealing in a client-server setting where it may be reasonable to believe there is an available server able to perform super-polynomial work to help compute on genomic data. Even when restricting all participants to polynomial resources, the result is a private protocol for the LCS problem achieving a sub-quadratic communication complexity: the first private LCS protocol whose efficiency improves on that of the generic solution using Yao's garbled circuit protocol. We have shown that the protocol and techniques extend (trivially, in some cases) to numerous other string algorithms useful for computing with genomic data. We anticipate this technique to be useful in achieving communication-efficient protocols for many other problems that have similarly-structured dynamic programming algorithms.

References

1. Are guarantees of genome anonymity realistic? `http://arep.med.harvard.edu/PGP/Anon.htm`, 2008.
2. CODIS: Combined DNA index system. `http://www.fbi.gov/hq/lab/html/codis1.htm`, 2008.
3. deCODE genetics. `http://www.decodegenetics.com/`, 2008.
4. The genomic privacy project. `http://privacy.cs.cmu.edu/dataprivacy/projects/genetic/`, 2008.
5. HapMap: International HapMap project. `http://www.hapmap.org/`, 2008.
6. Alfred V. Aho, Daniel S. Hirschberg, and Jeffrey D. Ullman. Bounds on the complexity of the longest common subsequence problem. *Journal of the ACM*, 23(1):1–12, 1976.
7. Bill Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In *Advances in Cryptology – Proceedings of Eurocrypt 2001*, pages 119–135, 2001.
8. Russ B. Altman and Teri E. Klein. Challenges for biomedical informatics and pharmacogenenomics. *Annual Review of Pharmacology and Toxicology*, 42:113–133, 2002.
9. V. L. Arlazarov, E. A. Dinic, M. A. Kronod, and I. A. Faradzev. On economic consruction of the transitive closure of a directed graph. *Doklady Akademii Nauk SSSR*, 194:487–488, 1970.
10. Mikhail J. Atallah, Florian Kerschbaum, and Wenliang Du. Secure and private sequence comparisons. In *Proceedings of the 2003 ACM Workshop on Privacy in the electronic society (WPES 2003)*, pages 39–44, 2003.

11. Justin Brickell and Vitaly Shmatikov. Privacy-preserving graph algorithms in the semi-honest model. In *Advances in Cryptology – ASIACRYPT*, pages 236–252, 2005.
12. Ran Canetti. Security and composition of multiparty cryptographic protocols. In *Journal of Cryptology*, volume 13, pages 143–202, 2000.
13. Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *Journal of the ACM*, 45(6):965–981, 1998. An earlier version appeared in FOCS 95.
14. Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, 2000.
15. Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *Public Key Cryptography*, pages 119–136, 2001.
16. Ivan Damgård and Mads Jurik. A length-flexible threshold cryptosystem with applications. In *Information Security and Privacy*, pages 350–364, 2003.
17. Matthew Franklin, Mark Gondree, and Payman Mohassel. Multi-party indirect indexing and applications. In *Advances in Cryptology – ASIACRYPT '07*, pages 283–297, 2007.
18. Matthew Franklin, Mark Gondree, and Payman Mohassel. Communication-efficient private protocols for longest common subsequence. In *The Cryptographer's Track at the RSA Conference (CT-RSA)*, 2009.
19. Campbell Fraser. *Subsequences and supersequences of strings*. PhD thesis, University of Glasgow, 1995.
20. Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology – Proceedings of Eurocrypt 2004*, pages 1–19, 2004.
21. Craig Gentry and Zulfikar Ramzan. Single-database private information retrieval with constant communication rate. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP 2005)*, pages 803–815, 2005.
22. Oded Goldreich. *Foundations of Cryptography*. Cambridge University Press, 2001.
23. Dan Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997.
24. Yuval Ishai, Tal Malkin, Martin J. Strauss, and Rebecca N. Wright. Private multiparty sampling and approximation of vector combinations. In *Proceedings of the 34th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 243–254, 2007.
25. Somesh Jha, Louis Kruger, and Vitaly Shmatikov. Towards practical privacy for genomic computation. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2008.
26. Eike Kiltz, Payman Mohassel, Enav Weinreb, and Matt Franklin. Secure linear algebra using linearly recurrent sequences. In *Proceedings of the Theory of Cryptography Conference (TCC 2007)*, pages 291–310, 2007.
27. Bradley Malin and Latanya Sweeney. Re-identification of dna through an automated linkage process. In *Proceedings of the AMIA Annual Symposium*, pages 423–427, 2001.
28. William J. Masek and Mike S. Paterson. A faster algorithm for computing string edit distances. *Journal of Computer and System Sciences*, 20:18–31, 1980.
29. Carlos Aguilar Melchor and Yves Deswarte. Single-database Private Information Retrieval schemes : overview, performance study, and usage with statistical databases. In *Privacy in Statistical Databases*, pages 257–265, 2006.

30. Moni Naor and Kobbi Nissim. Communication preserving protocols for secure function evaluation. In *STOC '01: Proceedings of the 33rd annual ACM Symposium on Theory of Computing*, pages 590–599, 2001.
31. Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *STOC '99: Proceedings of the 31st annual ACM Symposium on Theory of Computing*, pages 245–254, 1999.
32. Department of Health and Human Services. 45 CFR (Code of Federal Regulations), parts 160–164. Standards for privacy of individually identifiable health information, final rule. Federal Register: 67 (157): 53182-53273, August 12 2002.
33. The GPL Violations project. `http://gpl-violations.org/`, 2008.
34. Doug Szajda, Michael Pohl, Jason Owen, and Barry G. Lawson. Toward a practical data privacy scheme for a distributed implementation of the Smith-Waterman genome sequence comparison algorithm. In *Proceedings of the 2006 ISOC Network and Distributed System Security Symposium (NDSS 2006)*, pages 253–265, 2006.
35. Laszlo T. Vaszar, Mildred K. Cho, and Thomas A. Raffin. Privacy issues in personalized medicine. *Pharmacogenomics*, 4(2):107–112, 2003.
36. Andrew C. Yao. How to generate and exchange secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, pages 162–167, 1986.

A Private Backtracking Protocol

Here, we build a private protocol that emulates the strategy of some deterministic algorithm which recovers some LCS of A and B (or, with small modifications, strings which represent how the LCS is embedded in A or B). The protocol uses the data structures and basic tools of our LCS-length protocol. We note that the LCS-length protocol itself does not yield any extra information to the parties, since the output of any LCS-backtracking protocol will always leak the length of the LCS to the parties anyway.

The general idea of our LCS-backtracking protocol is to pre-compute (for each basic block) all possible backtracking paths through the basic block and the strings associated with these paths. We use "back-pointers" to later traverse through the L matrix and fetch the appropriate pre-computed data.

Before, Alice and Bob stored the recovered portions of the L matrix locally, and performed only local operations on it. Here, they will need to collaboratively backtrack through this table. Thus, the specific data structure organizing the shares they recover by querying the basic block function becomes relevant. Let H be the mn/t^2-sized table holding the data recovered by evaluating the basic block function in the LCS-length protocol. Specifically, let the data for the block starting at position (i, j) in the L matrix be stored at index $h(i, j)$ in H. That is, for Alice $H[h(i, j)]$ holds her shares of $x_1||x_2||x_3||x_4, \widetilde{E}(i, j), \widetilde{S}(i, j), I_1(i+t, j), I_2(i, j+t)$. To

backtrack through the H table, we will define an additional table H' such that

$$H'[h(i,j)||00] = h(i-1,j-1)$$
$$H'[h(i,j)||01] = h(i-1,j-1)$$
$$H'[h(i,j)||10] = h(i-1,j)$$
$$H'[h(i,j)||11] = h(i,j-1)$$

So, for the block in the L matrix whose data will be stored at index $h(i,j)$ in H, H' holds pointers to the three adjacent blocks that may be useful during backtracking. Alice and Bob will pre-compute and XOR share H'.

When backtracking through any $t \times t$ block, we will always start at one of $2t$ entry positions in $E(i,j), S(i,j)$, and exit through one of the $2t$ positions in $W(i,j), N(i,j)$. For each entry position k, we will pre-compute the exit position k'. Additionally, for each of these pre-computed paths, we can pre-compute the string (or embedding) corresponding to following this path.

Let $x = x_1||x_2||x_3||x_4$ be an index into the table defining the basic block function. Consider entering this basic block at the position represented by $k \in [1, 2t]$. If the deterministic backtracking strategy exits through the position represented by $k' \in [1, 2t]$ and produces the t-length string (or embedding) w (padded with a special ϵ empty string, as appropriate) then we define two additional tables, M, M' such that:

- $M[x||00||k] = k'$;
- $M[x||10||k] = u$, a $\log t$-length pointer such that $M'[x||u] = w$;
- $M[x||01||k] = z$, where $z \in \{0,1\}^2$ represents into which adjacent block backtracking might continue. Specifically,

$$z = \begin{cases} 01 & \text{if } k' \text{ represents } N(i,j)[1] \text{ (which is also } W(i,j)[1]) \\ 10 & \text{if } k' \text{ represents } N(i,j)[\lambda] \text{ for } 2 \leq \lambda \leq t \\ 11 & \text{if } k' \text{ represents } W(i,j)[\lambda] \text{ for } 2 \leq \lambda \leq t \end{cases}$$

With these data structures, backtracking will proceed in the following manner. Alice and Bob run the LCS-length protocol, filling out the H table. Initially, let k represent the index of the lower-left corner of the block, $i = h(m,n)$, $ans = \emptyset$, and $u = 0^{\log t}$:

1. Let $x = x_1||x_2||x_3||x_4$ be the share of the index stored at $H[i]$
2. Alice and Bob retrieve the following values (blinding, permuting, engaging in an SPBR protocol, in the appropriate way):
   $u \leftarrow M[x||10||k]$
   $w \leftarrow M'[x||u]$

$z \leftarrow M[x||01||k]$
$k \leftarrow M[x||00||k]$
$i \leftarrow H'[i||z]$

3. $ans \leftarrow w||ans$
4. Return to Step 1

Traditional backtracking ends when we move to position (i, j) in L and either $i = 0$ or $j = 0$, which will happen in at most $m + n$ steps. To avoid leaking information related to the number of rounds required for backtracking, we will always iterate the maximum number of times possible. Using the four Russians technique, this means iterating above for $m/t + n/t$ rounds.

In doing this, however, we must be a little more careful with our "pointer arithmetic." Specifically, we will construct a dummy location in H at index $h(0, -1)$. Then, we let $H'[h(i, j)||z] = h(0, -1)$ for $z \in \{0, 1\}^2$ if $i = 0$ or $j = 0$. At the dummy location $H[h(0, -1)]$,we will store a special input into the basic block function. We modify the basic block function table so that it is defined over this special (otherwise undefined) input sequence, it acts as a fixed point on this input, and every path through the block for this input is associated with the string ϵ^t during backtracking.

After $m/t + n/t$ rounds, Alice and Bob use their shares of the $m + n$ length string ans as input to a small garbed circuit that combines the shares, strips out the ϵ symbols, and outputs the final string to each party.

Claim. The above protocol is a private LCS-backtracking protocol, assuming the SPBR scheme is secure.

Proof. (sketch) The above security claim follows from the security of the share conversion protocols, our black-box use of SPBR, and from general composition theorems [12, 22]. Again, the Gentry-Ramzan PBR can be transformed into such a scheme using the appropriate transforms and computational assumptions.

A.1 Analysis

Our protocol for LCS-backtracking incurs all the costs of the LCS-length algorithm, with some additional overhead. The overhead is analyzed here. Above, M, M' are tables of size $O(2^t)$ with entries of length at most t, and H, H' are tables of size $O(mn/t^2)$ with entries of length $2t + 2t \log t$ and $O(\log(mn/t^2))$, respectively. As before, we can recover ℓ-bit entries from

a table of size D at a communication cost of $\log N$ by letting the database be of size $N = \ell D$ with the appropriate variant of Gentry-Ramzan. Thus, our per-round communication complexity is $O(\log(mn/t) + t)$. The protocol has total round complexity $m/t + n/t$. As a result, the total communication complexity is $O((m+n)\log(mn/t)) = O(n\log(n/t))$. This computational, round, and communication overhead is asymptotically less than the costs of the LCS-length protocol.