Privacy Preserving String Comparisons Based on Levenshtein Distance

Rane, S.; Sun, W.

TR2010-109 December 2010

Abstract

Alice and Bob possess strings x and y of length m and n respectively and want to compute the Levenshtein distance L(x, y) between the strings under privacy and communication constraints. The Levenshtein distance, or edit distance, has a dynamic programming formulation that solves a series of minimumfinding problems. Based on this formulation, there are known symmetric privacy-preserving protocols for the computation of L(x, y), in which the two parties incur equal protocol overhead. In this work, we propose an asymmetric two-party protocol in which a lightweight client Bob with a string y interacts with a single powerful server Alice containing string x in its database. We present a privacy-preserving minimum-finding protocol based on semantically secure homomorphic functions and additive secret sharing. This protocol is executed repeatedly, to enable private computation of the edit distance. Our protocol supports arbitrary finite insertion/deletion costs and a variety of substitution costs. While Alice requires similar effort as in previous approaches, the advantage is that Bob incurs far fewer ciphertext operations and transmissions, making the protocol well-suited for client-server querying applications.

IEEE International Workshop on Information Forensics and Security (WIFS)

Privacy Preserving String Comparisons Based on Levenshtein Distance

Shantanu Rane and Wei Sun

Mitsubishi Electric Research Laboratories, Cambridge, MA 02139

{rane,weisun}@merl.com

***Abstract*—Alice and Bob possess strings $\mathbf{x}$ and $\mathbf{y}$ of length m and n respectively and want to compute the Levenshtein distance $L(\mathbf{x}, \mathbf{y})$ between the strings under privacy and communication constraints. The Levenshtein distance, or edit distance, has a dynamic programming formulation that solves a series of minimum-finding problems. Based on this formulation, there are known symmetric privacy-preserving protocols for the computation of $L(\mathbf{x}, \mathbf{y})$, in which the two parties incur equal protocol overhead. In this work, we propose an asymmetric two-party protocol in which a lightweight client Bob with a string $\mathbf{y}$ interacts with a single powerful server Alice containing string $\mathbf{x}$ in its database. We present a privacy-preserving minimum-finding protocol based on semantically secure homomorphic functions and additive secret sharing. This protocol is executed repeatedly, to enable private computation of the edit distance. Our protocol supports arbitrary finite insertion/deletion costs and a variety of substitution costs. While Alice requires similar effort as in previous approaches, the advantage is that Bob incurs far fewer ciphertext operations and transmissions, making the protocol well-suited for client-server querying applications.**

I. INTRODUCTION

The problem of comparing two strings arises in many engineering systems, examples of which include keyword search with spelling mistakes, plagiarism detection, spoken query search, and comparison of gene sequences. Often, it is desired to find out if the strings are similar according to a suitable distance metric. One distance measure used to determine the extent to which two strings differ, is the Levenshtein distance [1] or edit distance. Informally, the edit distance between two strings represents the cost of making insertions, deletions and substitutions to transform one string into the other. A related problem is to determine the length of the longest subsequence that is common among two strings. Both these problems are solved using two very similar dynamic programming formulations [2].

The problem of computing edit distance becomes more complicated when the two strings are owned by two untrusting parties. In particular, private edit distance computation requires the parties to work through a dynamic programming solution without revealing any portion of the strings to each other. The literature contains a solution in which a "customer" Carol owns two strings $\mathbf{x}$ and $\mathbf{y}$ and outsources the computation to two untrusted servers [3], [4]. This is an elegant solution in which additive secret sharing is employed by Carol and the servers to prevent either server from finding out anything about $\mathbf{x}$ and $\mathbf{y}$. The servers are symmetric with respect to the computation and communication overhead, and are assumed not to collude with each other. To our knowledge, the only other work on this problem is the three-party protocol of Inan *et al.* [5] in which Alice and Bob enlist the help of a semi-honest third party. In this solution, the third party finds out nothing about Alice's string $\mathbf{x}$ and Bob's string $\mathbf{y}$ but does find out the edit distance between every substring of $\mathbf{x}$ and $\mathbf{y}$.

We tackle the privacy-preserving edit distance computation from a different perspective. Specifically, there is a client Bob who wishes to compute the edit distance between his string $\mathbf{y}$ and another string $\mathbf{x}$ which is stored by a database server, Alice. Bob has limited computation capability and upload bandwidth while Alice has a more powerful computer with higher upload bandwidth. Therefore our protocol is designed to be asymmetric in terms of the computational and communication load that it places on Alice and Bob. This asymmetric scenario is more common in practice; a lightweight end-user wants to perform a private query against a more powerful database server. Moreover, since there are only two parties, the problem of collusion need not be considered. We assume that the parties are semi-honest, i.e., they will follow the rules of the protocol but will attempt to glean as much information as possible from the data that is available to them at each intermediate step.

The main tools used in our protocols are additive secret sharing and semantically secure additively homomorphic encryption. Semantic security ensures that repeated encryptions of the same plaintext result in different ciphertexts. Usually, semantic security is accomplished by using a random parameter at the time of encryption; the ciphertext differs depending on the value of the random parameter, which is not required at the time of decryption. We will point out wherever semantic security is useful in our protocols, but the random parameter will be omitted to avoid notational clutter. The protocol works with any of the published semantically secure additive homomorphic cryptosystems of Paillier [6], Damgård-Jurik [7] and Benaloh [8]. We have previously used homomorphic encryption to develop efficient privacy-preserving protocols for computation of Hamming distance, squared error [9] and absolute error [10] between two sequences of equal length. In this work, the focus is on comparing strings not necessarily of equal length.

The remainder of this paper is organized as follows: Sec-

WIFS'2010, December 12-15, 2010, Seattle, WA, USA. 978-1-4244-9080-6/10/\$26.00 ©2009 IEEE.

tion II gives a formal definition of edit distance and describes a dynamic programming solution to compute it. Section III proposes a privacy-preserving minimum-finding protocol that will be used as a primitive in the edit distance computation. Section IV describes privacy-preserving protocols for efficiently computing some commonly used substitution costs. In Section V, the full privacy-preserving edit distance protocol is described, using repeated invocations of the minimum-finding and substitution cost protocols. Finally, Section VI extends the edit distance protocol to the problem of measuring the similarity between two strings without sharing them.

II. String Edit Distance

Consider a finite alphabet set $\mathcal{A}$ whose elements will be used to construct strings. Let $\mathcal{Z}_I, \mathcal{Z}_D$ and $\mathcal{Z}_S$ be finite sets whose elements are finite integers. Let the function $I\colon \mathcal{A} \mapsto \mathcal{Z}_I$ be the insertion cost function, i.e., $I(a)$ is the cost of inserting the element $a \in \mathcal{A}$ into a given string. Similarly, define the deletion cost function as $D\colon \mathcal{A} \mapsto \mathcal{Z}_D$ so that $D(a)$ is the cost of deleting the element a from a given string. Finally, define the substitution cost function $S\colon \mathcal{A} \times \mathcal{A} \mapsto \mathcal{Z}_S$ so that, for $a, b \in \mathcal{A}$, $S(a,b)$ is the cost of replacing the element a in a given string by the element b.

Now consider two strings of length m and n, denoted by $\mathbf{x} \in \mathcal{A}^m$ and $\mathbf{y} \in \mathcal{A}^n$. Consider the sequence of insertion, deletion and substitution operations needed to transform $\mathbf{x}$ into $\mathbf{y}$ and the corresponding aggregate cost of the transformation. For a given pair of strings, the sequence of operations and the aggregate cost is not unique in general.

Definition 1: The string edit distance or Levenshtein distance is defined as the minimum aggregate cost of transforming $\mathbf{x}$ into $\mathbf{y}$.

Definition 2: For all $a, b \in \mathcal{A}$, let $I(a) = D(a) = 1$, $S(a,b) = 1$ when $a \neq b$, and $S(a,a) = 0$. Then, the edit distance is defined as the minimum number of insertions, deletions and substitutions required to convert $\mathbf{x}$ into $\mathbf{y}$.

A. Dynamic Programming Formulation

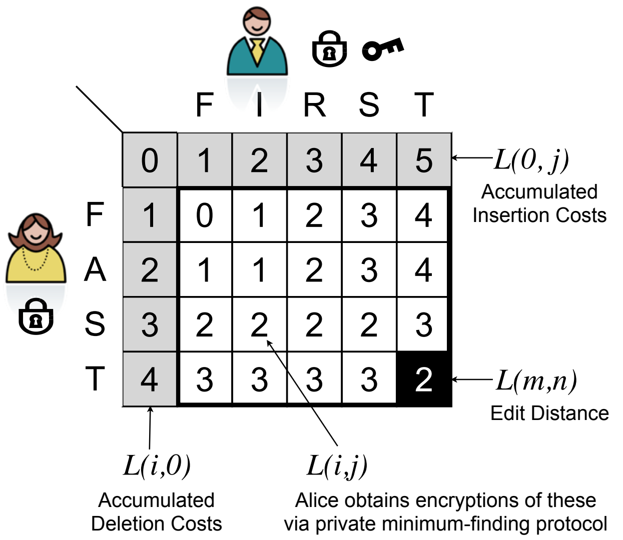
When $\mathbf{x} = x_1 x_2 ... x_m$ and $\mathbf{y} = y_1 y_2 ... y_n$ are clear from the context, we use $L(i,j)$ to denote the edit distance between the two substrings $x_1 x_2 ... x_i$ and $y_1 y_2 ... y_j$. Thus, $L(\mathbf{x}, \mathbf{y})$ can also be written $L(m,n)$. The problem of finding the edit distance can be solved in $O(mn)$ time by a well-known dynamic programming formulation [2] which we will adapt to the two-party scenario of this paper.

Fix $L(0,0) = 0$. Define for $1 \le i \le m, 1 \le j \le n$:

$$L(i,0) = \sum_{k=1}^{i} I(x_k) \quad \text{and} \quad L(0,j) = \sum_{k=1}^{j} D(y_k) \qquad (1)$$

Then, the edit distance $L(m,n)$ is defined by the following recurrence relation for $1 \le i \le m, 1 \le j \le n$:

$$L(i,j) = \min \left\{ \begin{array}{c} L(i-1,j) + D(y_j), \\ L(i,j-1) + I(x_i), \\ L(i-1,j-1) + S(x_i, y_j) \end{array} \right\} \qquad (2)$$



F I R S T
F A S T
0 1 2 3 4 5
1 0 1 2 3 4
2 1 1 2 3 4
3 2 2 2 2 3
4 3 3 3 3 2
L(0, j)
Accumulated Insertion Costs
L(m,n)
Edit Distance
L(i,0)
Accumulated Deletion Costs
L(i,j)
Alice obtains encryptions of these via private minimum-finding protocol

Fig. 1. An example of edit distance computation, in which Alice owns the string FAST and Bob owns the string FIRST. In our asymmetric setup, only Bob, the end-user possesses the decryption key for additively homomorphic encryption. Alice, the server, computes the encryptions of $L(i,j)$ via an interactive protocol that will be described in the subsequent sections.

The remainder of this paper describes how Alice and Bob can compute $L(i,j)$ in a privacy-preserving manner. A high-level description of the protocol is as follows: Alice, the more powerful computer, engages in a minimum-finding protocol with Bob, at the end of which, she obtains encryptions of $L(i,j)$. She is not able to decrypt these values and therefore is unable to extract any information about Bob's string $\mathbf{y}$. The protocol does not allow Bob to discover $L(i,j), 1 \le i < m, 1 \le j < n$, thereby keeping Alice's string concealed from Bob. At the end of the protocol, Alice sends the encryption of $L(m,n)$ to Bob, who decrypts it and obtains the desired edit distance. If necessary, Bob can then share this value with Alice. Fig. 1 shows an example of edit distance calculation under this framework. The example shown in the figure uses unit insertion costs, unit deletion costs, and indicator function substitution costs as in Definition 2

Our method is motivated by the asymmetry in the computational capabilities of Alice and Bob, and therefore differs from the symmetric approach of [3], [4] in which Alice and Bob compute additive shares of $L(i,j)$ for all i, j. Before describing the string edit distance protocol, we first describe a privacy-preserving minimum-finding protocol for two parties. The minimum-finding protocol is a primitive that will be invoked repeatedly in order to find the string edit distance.

III. Privacy-Preserving Minimum Finding

Let (k_e, k_d) be the encryption/decryption key pair for a semantically secure additively homomorphic public key cryptosystem. Denote the encryption and decryption functions by $\xi(\cdot)$ and $\xi^{-1}(\cdot)$ respectively. The additively homomorphic property ensures that $\xi(m_1)\xi(m_2) = \xi(m_1 + m_2)$ and $\xi(m_1)^{m_2} = \xi(m_1 m_2)$ for integer messages m_1, m_2. Further, consider a vector of integers $\mathbf{z} \in \mathbb{Z}^N$. Let $\xi(\mathbf{z}) = (\xi(z_1), \xi(z_2), \cdots, \xi(z_N))$.

Inputs: Alice has the public encryption key k_e and $\xi(\mathbf{z})$, Bob has the key-pair (k_e, k_d)
Outputs: Alice obtains $\xi(\min_{1\le i\le N} z_i)$. Bob obtains nothing.
Conditions: Bob should not discover any of the z_i. Neither Alice nor Bob should discover the index $\arg\min_i z_i$

The steps of the protocol are as follows:

1) Alice generates a permutation π on the set $\{1, 2, \cdots, N\}$ and obtains an encrypted vector $\xi(\mathbf{v}) = \pi(\xi(\mathbf{z}))$.
2) Alice chooses an integer $g > 0$ and generates a matrix $\mathbf{G} \in \mathbb{Z}^{N\times N}$ as follows:
$$\mathbf{G} = \begin{bmatrix} g+g_1 & g_2 & g_3 & \cdots & g_N \\ g_1 & g+g_2 & g_3 & \cdots & g_N \\ g_1 & g_2 & g+g_3 & \cdots & g_N \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_1 & g_2 & g_3 & \cdots & g+g_N \end{bmatrix}. \quad (3)$$
Let $\mathbf{w} = \mathbf{G}\mathbf{v}$. Then, for any $1 \le i, j \le N$, $w_i - w_j = g(v_i - v_j)$, which means that for $g > 0$, $\mathbf{G}$ is an order-preserving map[1]. In other words, $v_i \le v_j$ iff $w_i \le w_j$. Using the properties of additive homomorphic encryption, Alice determines $\xi(\mathbf{w}) = \xi(\mathbf{G}\mathbf{v})$.
3) Alice generates a vector $\mathbf{a} \in \mathbb{Z}^N$ of random integers. Using homomorphic properties, she obtains $\xi(\mathbf{w} - \mathbf{a})$ and sends it to Bob.
4) Bob obtains $\mathbf{b} = \mathbf{w} - \mathbf{a}$ by element-wise decryption. Notice that, at this stage Alice has $\mathbf{a}$ and Bob has $\mathbf{b}$, which are just additive shares of $\mathbf{w}$. Therefore $w_i \le w_j$ iff $a_i + b_i \le a_j + b_j$ iff $a_i - a_j \le b_j - b_i$.
5) Alice constructs a new vector $\mathbf{a}_\triangle$ whose elements are the pairwise differences of the elements of $\mathbf{a}$. Thus, $\mathbf{a}_\triangle = (a_1 - a_2, a_1 - a_3, \cdots, a_1 - a_N, a_2 - a_3, a_2 - a_4, \cdots, a_2 - a_N, \cdots, a_{N-1} - a_N)$. Similarly, Bob obtains a vector $\mathbf{b}_\triangle = (b_1 - b_2, b_1 - b_3, \cdots, b_1 - b_N, b_2 - b_3, b_2 - b_4, \cdots, b_2 - b_N, \cdots, b_{N-1} - b_N)$. Observe that $\mathbf{a}_\triangle$ and $\mathbf{b}_\triangle$ both contain $\binom{N}{2}$ elements.
6) Alice chooses an integer η at random over $[-g, g]$. She generates a vector $\boldsymbol{\eta} = (\eta_{ij})_{1\le i<j\le N} \in \mathbb{Z}^{\binom{N}{2}}$ such that $-g \le \eta_{ij} \le g$ and $\sum_{1\le i<j\le N} \eta_{ij} = \eta$. She sends the vector $\mathbf{a}_\triangle - \boldsymbol{\eta}$ to Bob.
7) Bob compares the corresponding elements of $\mathbf{a}_\triangle - \boldsymbol{\eta}$ and $-\mathbf{b}_\triangle$. For, any element η_{ij} in the noise vector $\boldsymbol{\eta}$, Bob determines for $1 \le i < j \le N$ whether or not $a_i - a_j - \eta_{ij} \le b_j - b_i$, which is equivalent to determining whether or not $g(v_i - v_j) - \eta_{ij} \le 0$ by the construction of the $\mathbf{G}$ matrix.
Since $-g \le \eta_{ij} \le g$, Bob knows that $a_i - a_j - \eta_{ij} \gtrless b_j - b_i$ if and only if $v_i \gtrless v_j$. Thus, the order is preserved when $v_i \ne v_j$. If $v_i = v_j$, then $a_i - a_j - \eta_{ij}$ may be greater or less than $b_j - b_i$ depending on the value of η_{ij} that was randomly chosen by Alice[2]. Nevertheless, this perturbation still allows Bob to obtain $\alpha = \arg\min_{1\le i\le N} v_i$ without knowing v_i or w_i. He sends $\xi(\alpha)$ to Alice.
8) For $1 \le i \le N$, Alice choses integers r and β_i at random and uses homomorphic properties to compute $\xi(\beta_i(i - \alpha) + v_i + r)$. She sends the resulting vector of N encrypted entries to Bob.
9) Bob decrypts the entry corresponding to the index $i = \alpha$ to obtain $v_\alpha + r$. He then re-encrypts this term and sends it back to Alice.
10) Due to the semantic security property Alice cannot determine the π-permuted index of the minimum, i.e., α by looking at the ciphertext. She removes the random noise r via $\xi(v_\alpha + r)\xi(-r) = \xi(v_\alpha) = \xi(\min_{1\le i\le N} v_i) = \xi(\min_{1\le i\le N} z_i)$

Security and Privacy Analysis: In Steps 1 and 2, Alice operates in the encrypted domain, and therefore finds out nothing about $\mathbf{z}$. In Steps 3-6, Bob operates only on additive shares of $\mathbf{w}$ and $\mathbf{a}_\triangle$. Therefore, he does not discover anything about $\mathbf{w}$ and $\mathbf{a}_\triangle$. In Step 7, Bob can obtain
$$\delta_{ij} \overset{def}{=} (a_i + b_i) - (a_j + b_j) - \eta_{ij} = g(v_i - v_j) - \eta_{ij}$$
where $1 \le i < j \le N$. There are $\binom{N}{2}$ values of the noise terms η_{ij} that have been chosen by Alice but are unknown to Bob. This makes it impossible for Bob to discover $v_i - v_j$, let alone v_i and v_j. Bob could try another strategy, which is to add all the $\delta_{i,i+1}$ and obtain
$$\sum_{1\le i<N} \delta_{i,i+1} = g(v_1 - v_N) - \sum_{1\le i<N} \eta_{i,i+1}.$$
However, he does not know $\sum_{1\le i<N} \eta_{i,i+1}$. Thus, Bob is still unable to discover whether $v_1 = v_N$ or not. In Steps 8 and 9, Bob discovers only an additive share of the minimum, given by $v_\alpha + r$, but cannot discover v_α since he does not know the value of r. Finally, in Step 10, Alice does not know the index α of the minimum owing to the semantic security of the homomorphic function. Specifically, when Bob re-encrypts $v_\alpha + r$, he chooses a random encryption parameter which obfuscates the index of the minimum from Alice. Thus, in the semi-honest setting, Alice finds out the encryption of the minimum without finding out either the value of the minimum or the index of the minimum. Bob performs the role of a helper in this protocol by performing decryptions wherever necessary, but does not discover any information about the z_i. Moreover, other than decryptions, all other computations at Bob's end are in the plaintext domain and therefore incur a very small overhead.

Table I shows the cost in terms of computation and communication incurred by the two parties. For comparison, we also show the cost for a symmetric realization of the minimum finding protocol [4], in which Alice and Bob obtain additive shares of the minimum. According to our reading of [4], the protocol preserves privacy in all cases, except for scenarios in which two or more elements in the vector being queried are equal. Specifically, Alice and Bob don't explicitly find out

[1]It can be shown that $\mathbf{G}$ is the only map that is order-preserving.

[2]Bob essentially observes random perturbations of $w_i - w_j$. Therefore, the probability of ties in Step 7 is vanishingly small, but even if a tie occurs, Bob chooses any one of the tied indices as the π-permuted index of the minimum.

Quantity	Alice (Server) Proposed	Bob (End-User) Proposed	Symmetric [4] reveals equalities	[4] without revealing equalities
# Encryptions	$O(N)$	$O(1)$	$O(N)$	$O(N^2)$
# Decryptions	None	$O(N)$	$O(N)$	$O(N^2)$
# Ciphertext Transmissions	$O(N)$	$O(1)$	$O(N)$	$O(N^2)$
# Adds and Multiplies in ciphertext	$O(N^2)$	None	$O(N)$	$O(N)$

TABLE I
THE END-USER INCURS LOW COMPUTATIONAL AND COMMUNICATION COMPLEXITY IN THE PROPOSED PRIVACY-PRESERVING MINIMUM-FINDING PROTOCOL. THE OVERHEAD IN A SYMMETRIC REALIZATION OF MINIMUM-FINDING IS PROVIDED FOR COMPARISON.

the value of these elements, but, with the simple minimum finding protocol required at the end of the protocol, they will find out whether or not some elements in the vector $\mathbf{z}$ are equal to others. This can be remedied by using a secure millionaire protocol instead of the simple minimum finding protocol in the final stage, at the cost of increased complexity. Therefore, the costs for the symmetric scheme in the last column of Table I assume that a millionaire protocol is used $\binom{N}{2}$ times. In the protocol that we presented above, the noise vector $\boldsymbol{\eta}$ prevents Bob from knowing whether any of the elements were equal or not. Alice operates only on encrypted data, so she remains oblivious of the presence of equalities.

IV. Private Substitution Cost Protocols

Recall that, in the two party setup, one party (say Alice) knows the insertion costs $I(x_i), 1 \leq i \leq m$, while the other party knows the deletion costs $D(y_j), 1 \leq j \leq n$. The substitution costs, however, must be computed interactively using a privacy preserving protocol. In general, arbitrary substitution cost functions can be computed with privacy using Oblivious Transfer (OT) protocols. However, using additive homomorphisms, some commonly used substitution costs can be computed more efficiently than by using the generalized OT-based framework for secure function evaluation.

Inputs: Alice has a symbol $a \in \mathcal{A}$, and a public key k_e for the homomorphic encryption function $\xi(\cdot)$. Bob has a symbol $b \in \mathcal{A}$, and the public-private key pair (k_e, k_d).
Outputs: Alice obtains $\xi(S(a,b))$.
Conditions: Alice should not discover b. Bob should not discover a

A. Absolute distance cost: $S(a,b) = |a-b|$

Note that $S(a,b) = -\min(a-b, b-a)$.The steps of the protocol are as follows:

1) Bob sends $\xi(b)$ to Alice.
2) Alice obtains $\xi(a-b)$ and $\xi(b-a)$ using the additive homomorphic property.
3) Then, Alice and Bob execute the privacy-preserving minimum-finding protocol described in Section III. At the end of the protocol, Alice obtains $\xi(-\min(a-b, b-a)) = \xi(S(a,b))$ as required.

Quantity	Alice	Bob
Encryptions	None	$O(\max u)$
Decryptions	None	None
Ciphertext Transmissions	None	$O(\max u)$
Adds & Multiplies in ciphertext	$O(a^{\max t})$	None

TABLE II
OVERHEAD FOR POLYNOMIAL SUBSTITUTION COST PROTOCOL. THIS PROTOCOL IS IMPRACTICAL FOR LARGE t OWING TO VERY HIGH COMPUTATIONAL COMPLEXITY AT THE SERVER.

The privacy, computational cost and communication overhead of this protocol derives from the minimum finding protocol of Section III.

B. Polynomial cost: $S(a,b) = \sum_{t,u} \gamma_{t,u} a^t b^u$ with $\gamma_{t,u} \in \mathbb{Z}$, $t,u \in \mathbb{Z}^+$

This function includes cost functions of the form $S(a,b) = (a-b)^t, t \in \mathbb{Z}^+$. The steps of the protocol are as follows:

1) Bob sends $\xi(b^u)$ to Alice for all the terms in $S(a,b)$ that contain positive powers of b.
2) Alice computes

$$\prod_{t,u} \xi(b^u)^{\gamma_{t,u} a^t} = \prod_{t,u} \xi(\gamma_{t,u} a^t b^u) = \xi\left(\sum_{t,u} \gamma_{t,u} a^t b^u\right)$$

The privacy of this protocol is based on the fact that Alice operates solely in the encrypted domain and cannot discover Bob's inputs. The protocol overhead in terms of encryptions, decryptions and ciphertext transmissions is given in Table II.

C. Indicator function cost: $S(a,b) = (1 - \mathbf{1}_{\{a=b\}})C, C \in \mathbb{Z}$

This is a commonly used substitution cost. The conventional implementation, based on oblivious transfer, would incur $O(|\mathcal{A}|)$ complexity from Bob. Since Bob is an end-user in our framework, we present below an alternative protocol which has $O(1)$ complexity for Bob. The steps of the protocol are:

1) Alice constructs an indicator vector $\mathbf{E}^{(a)} = (E_i^{(a)})$ of length $|\mathcal{A}|$ containing all 0's except a 1 at position a, that is, $E_i^{(a)} = 1$ if $i = a$, and 0 otherwise.
2) Bob sends $\xi(b)$ to Alice.
3) For $1 \leq i \leq |\mathcal{A}|$, Alice uses the properties of additive homomorphisms to compute $\xi(\beta_i(i-b) + (1-E_i^{(a)})C + r)$, where β_i and r are integers chosen at random from an appropriately large integer field. She sends these encryptions to Bob in the correct order.
4) From the received encryptions, Bob discards all except the b^{th} term and decrypts it to obtain $(1-E_b^{(a)})C + r$. Then, he re-encrypts this value and sends $\xi((1-E_b^{(a)})C + r)$ back to Alice.
5) Due to the semantic security property, Alice does not know which of the encrypted transmissions Bob returned to her. She just removes the noise term and obtains $\xi((1-E_b^{(a)})C + r)\xi(-r) = \xi((1-E_b^{(a)})C) = \xi((1-\mathbf{1}_{\{a=b\}})C) = \xi(S(a,b))$ as desired.

The privacy of this protocol derives from the privacy of 1-out-of-$|\mathcal{A}|$ OT and from the semantic security of homomorphic encryption which ensures that Alice cannot distinguish

Quantity	Alice	Bob
Encryptions	$O(\|\mathcal{A}\|)$	$O(1)$
Decryptions	None	$O(1)$
Ciphertext Transmissions	$O(\|\mathcal{A}\|)$	$O(1)$
Adds & Multiplies in ciphertext	$O(\|\mathcal{A}\|)$	None

TABLE III
OVERHEAD FOR INDICATOR FUNCTION SUBSTITUTION COST PROTOCOL.

between encryptions of 1's and 0's. The protocol overhead is given in Table III.

V. PRIVACY PRESERVING EDIT DISTANCE PROTOCOL

The protocol for privacy-preserving computation of edit distance can now be implemented using the minimum finding and substitute cost protocols presented in Section III and Section IV as primitives. Following the dynamic programming formulation for computing edit distance, the protocol enables Alice to calculate the encryption of the matrix $\xi(\mathbf{L}) = (\xi(L(i,j)))$. Bob finds out nothing about the matrix except the last entry $L(m,n) = L(\mathbf{x},\mathbf{y})$.

Inputs: Alice has string $\mathbf{x}$, and a public key k_e for the homomorphic encryption function $\xi(\cdot)$. Bob has string $\mathbf{y}$, and the public-private key pair (k_e, k_d).
Outputs: Bob obtains the edit distance $L(m,n)$.
Conditions: Alice will not discover any information about $\mathbf{y}$. Bob will not discover any information about $\mathbf{x}$

The steps of the protocol are as follows:

1) Alice keeps a table of $I(x_i)$ and $L(i,0), 1 \le i \le m$. Bob keeps a table of $D(y_j)$ and $L(0,j), 1 \le j \le n$.
2) Alice obtains the encryption of substitution cost $S(x_1,y_1)$, i.e., $\xi(S(x_1,y_1))$. In general, the encryption of any function $S(\cdot,\cdot)$ can be evaluated by means of oblivious transfer (OT). For some commonly used substitution cost functions, however, the protocols presented in Section IV can be used.
3) For $i = 1,2,\cdots,m$
   For $j = 1,2,\cdots,n$, with $(i,j) \neq (1,1)$
   (a) Bob transmits $\xi(D(y_j))$ to Alice.
   (b) Alice and Bob use a substitution cost protocol, at the end of which, Alice gets $\xi(S(x_i,y_j))$.
   (c) Using homomorphic encryption, Alice computes the encryptions of the three terms inside the curly brackets in (2), i.e. $\xi(L(i-1,j)+D(y_j))$, $\xi(L(i,j-1)+I(x_i))$, $\xi(L(i-1,j-1)+S(x_i,y_j))$.
   (d) Then, Alice and Bob execute the privacy-preserving minimum-finding protocol of Section III. At the end of the protocol, Alice has $\xi(L(i,j))$. Note that, Alice choses new parameters $\mathbf{G}$ and η for every execution of the minimum-finding protocol of Section III. As the parameters change with every instance, Bob cannot discover their values or use statistical analysis to guess the values $L(i,j)$ in the edit distance matrix.
4) Alice sends $\xi(L(m,n))$ to Bob, who decrypts it. This is the required edit distance.

The security derives from that of the proposed minimum finding and substitute cost protocols. At the end, Alice is left with the encrypted $L(i,j)$, while Bob discovers nothing about the matrix and Alice's data except for the edit distance. Since there are only two parties in the protocol, the question of collusions does not arise. The dynamic programming framework, combined with the requirement that Alice cannot decrypt any data, allows parallelization while preserving privacy. Suppose that Alice has a cloud of server nodes at her disposal, each with access to $\mathbf{x}$ and to the encryptions $\xi(L(i,j))$ as they become available according to privacy-preserving realizations of (2). The dynamic programming formulation ensures that if one server has obtained $\xi(L(i,j))$, then another server can independently obtain $\xi(L(i+1,j'))$ for $j' = 1,2,\cdots,j$. This reduces the computational load on any single server node in the cloud, and reduces the overall time required for Bob to obtain $L(m,n)$. As the server nodes are unable to decrypt Bob's transmissions, or to decrypt the $L(i,j)$, interaction amongst the nodes reveals no information about Bob's string.

VI. SIMILARITY MEASUREMENT WITH PRIVACY

The dynamic programming formulation for edit distance can be extended to compute the similarity between sequences $\mathbf{x}$ and $\mathbf{y}$. These extensions accommodate a variety of similarity measures, including a widely used metric: the length of the longest subsequence common to $\mathbf{x}$ and $\mathbf{y}$. These extensions necessitate small changes in the privacy-preserving edit distance protocol that are described below.

Let $\mathcal{A}' = \mathcal{A} \cup \{\_\}$, where $\mathcal{A}$ is the alphabet set defined earlier and $\{\_\}$ denotes a space. The space symbol is used to construct *alignments* between $\mathbf{x}$ and $\mathbf{y}$, i.e., if $\mathbf{x}$ and $\mathbf{y}$ have different lengths, then an appropriate number of spaces can be inserted into one or both strings, so that the augmented strings $\mathbf{x}'$ and $\mathbf{y}'$ are of equal length. The alignments only involve insertion of spaces and they necessarily preserve the relative order of the elements in each string, for e.g.,

$$\begin{array}{ccc} \text{F A T E S} & \rightarrow & \text{F - A T E S} \\ \text{J E A N S} & \rightarrow & \text{J E A N - S} \end{array}$$

Clearly, many such alignments are possible, and each alignment is associated with a score as follows: Akin to the substitution cost function in Section II, define the pair-wise score function $S\colon \mathcal{A}' \times \mathcal{A}' \mapsto \mathcal{Z}_S$ so that, for $a,b \in \mathcal{A}'$, $S(a,b)$ is the score for elements a and b located in the same locations in the first and second augmented string respectively. To prevent insertion of redundant spaces, force $S(\_,\_) = -\infty$, which is equivalent to imposing that $\mathbf{x}'$ and $\mathbf{y}'$ can never have a space at the same position. The *value* of an alignment is given by the sum of the scores for each pair (x'_i, y'_i) in the augmented strings.

Definition 3: The similarity between $\mathbf{x}$ and $\mathbf{y}$ is defined as the maximum of the values of all possible alignments.

One application of similarity measurement is to find the length of the longest subsequence common to $\mathbf{x}$ and $\mathbf{y}$. A subsequence is defined as a subset of the elements of a string

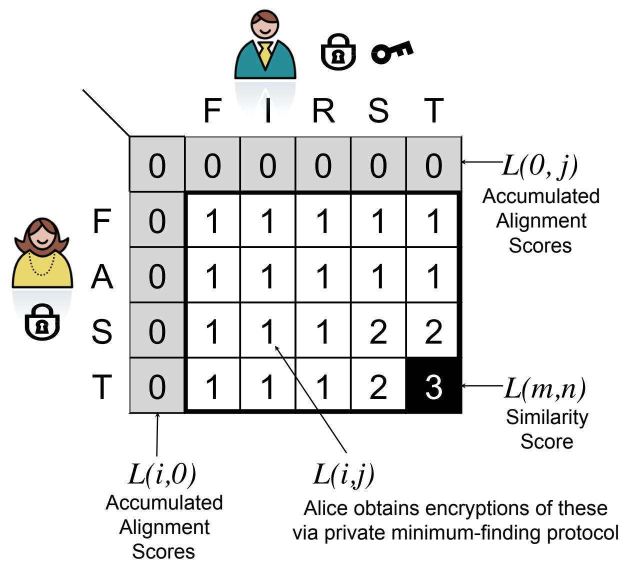

F I R S T
F A S T
0 0 0 0 0 0
0 1 1 1 1 1
0 1 1 1 1 1
0 1 1 1 2 2
0 1 1 1 2 3
L(0, j)
Accumulated Alignment Scores
L(m,n)
Similarity Score
L(i,0)
Accumulated Alignment Scores
L(i,j)
Alice obtains encryptions of these via private minimum-finding protocol

Fig. 2. An example of similarity computation, in which Alice owns the string FAST and Bob owns the string FIRST. As before, only Bob possesses the decryption key for additively homomorphic encryption. Alice computes the encryptions of $L(i, j)$ via an interactive protocol identical to the edit distance protocol. Only the encryption of the final value $L(m, n)$ is sent to Bob, who decrypts it to obtain the similarity, which in this case, is the length of the longest common subsequence, "FST".

in the original order. For e.g., ABLE is a subsequence of WARBLER, and the longest common subsequence of WARBLER and WEAVER is WAER.

Theorem 1: [2] For all $a, b \in \mathcal{A}'$, and $S(a, b) = \mathbf{1}_{\{a=b\}}$, the similarity gives the length of the longest common subsequence of $\mathbf{x}$ and $\mathbf{y}$.

Now, let us convert the problem of privacy-preserving computation of edit distance into the problem of privacy-preserving computation of similarity. As in Section II, let $\mathbf{x} = x_1x_2...x_m$ and $\mathbf{y} = y_1y_2...y_n$ be the two strings. Once again, use $L(i, j)$ to denote the similarity between the two substrings $x_1x_2...x_i$ and $y_1y_2...y_j$. Instead of the insertion cost function from Section II, define $I(a) = S(a, \_)$ for $a \in \mathcal{A}$. Similarly, instead of the deletion cost function, define $D(b) = S(\_, b)$ for $b \in \mathcal{A}$. With these new definitions, set up the base conditions exactly as in (1). Then, the similarity $L(m, n)$ is defined by the following recurrence relation for $1 \leq i \leq m, 1 \leq j \leq n$:

$$L(i,j) = \max \left\{ \begin{array}{c} L(i-1,j) + D(y_j), \\ L(i,j-1) + I(x_i), \\ L(i-1,j-1) + S(x_i,y_j) \end{array} \right\} \quad (4)$$

The desired similarity between the strings is $L(m, n)$. Thus, once the score function $S(\cdot, \cdot)$ has been specified and the insertion and deletion cost functions are modified as above, the edit distance formulation in (2) and similarity formulation in (4) are nearly identical. In fact, the only difference is that the former requires minimum-finding whereas the latter requires maximum-finding under privacy constraints. These operations both use the same protocol because $\max_i z_i = -\min_i(-z_i)$.

Fig. 2 shows an example in which Alice and Bob compute the similarity in a privacy-preserving manner. The example uses the indicator function score, i.e., $S(a, b) = \mathbf{1}_{\{a=b\}}$ as in Theorem 1 above, therefore the obtained similarity is the length of the longest subsequence common to Alice's and Bob's strings. For this problem, the score function $S(x_i, y_j)$ for $1 \leq i \leq m, 1 \leq j \leq n$ is determined using a privacy-preserving protocol nearly identical to the indicator function substitution cost protocol from Section IV-C. Note that the edit distance between FAST and FIRST is 2 while the length of the longest common subsequence is 3.

VII. Conclusions

This paper presented a two party protocol for privacy-preserving computation of the edit distance between strings held by two parties. The protocol is asymmetric in the sense that one of the parties has limited computing resources while the other is a more powerful database server. To compute the edit distance, Alice and Bob execute several instances of a privacy preserving minimum-finding protocol that requires fewer encryptions and encrypted transmissions from the lightweight customer (Bob). The protocol supports arbitrary finite insertion and deletion costs. The use of semantically secure additively homomorphic encryption functions has been shown to allow efficient computation of various useful substitution cost functions. The privacy-preserving protocol extends in a straightforward manner to similar problems in string comparison that admit a dynamic programming formulation. In particular, an extension is described in which the customer Bob interacts with the server Alice to find the length of the longest subsequence common to their strings.

References

[1] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, Feb. 1966.

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

[3] M. Atallah and J. Li, "Secure outsourcing of sequence comparisons," *International Journal of Information Security*, vol. 4, no. 4, pp. 277–287, Oct. 2005.

[4] M. Atallah, F. Kerschbaum, and W. Du, "Secure and private sequence comparisons," in *Proceedings of the 2003 ACM Workshop on Privacy in the Electronic Society*, Washington, DC, Oct. 2003, pp. 39–44.

[5] A. Inan, S. Kaya, Y. Saygin, E. Savas, A. Hintoglu, and A. Levi, "Privacy preserving clustering on horizontally partitioned data," *Data and Knowledge Engineering*, vol. 63, no. 3, pp. 646–666, Dec. 2007.

[6] P. Paillier, "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes," in *Advances in Cryptology, EUROCRYPT 99*. 1999, vol. 1592, pp. 233–238, Springer-Verlag, Lecture Notes in Computer Science.

[7] I. Damgård and M. Jurik, "A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System," in 4^{th} *International Workshop on Practice and Theory in Public Key Cryptosystems*, Cheju Island, Korea, Feb. 2001, pp. 119–136.

[8] J. Benaloh, "Dense Probabilistic Encryption," in *Proceedings of the Workshop on Selected Areas of Cryptography*, Kingston, ON, Canada, May 1994, pp. 120–128.

[9] S. Rane, W. Sun, and A. Vetro, "Secure Distortion Computation in the Encrypted Domain Using Homomorphic Encryption," in *Proc. IEEE International Conference on Image Processing*, Cairo, Egypt, Nov. 2009, pp. 1485–1488.

[10] S. Rane, W. Sun, and A. Vetro, "Privacy Preserving Approximation of L1 Distance for Multimedia Applications," in *Proc. IEEE International Conference on Multimedia and Expo (ICME), To Appear*, Singapore, July. 2010.