Secure Cloud Computing for Pairwise Sequence Alignment

Sergio Salinas
Department of Electrical Engineering and Computer Science
Wichita State University
salinas@cs.wichita.edu

Pan Li
Department of Electrical Engineering and Computer Science
Case Western Reserve University
lipan@case.edu

ABSTRACT

Today's massive amount of biological sequence data has the potential to rapidly advance our understanding of life's processes. However, since analyzing biological sequences is a very expensive computing task, users face a formidable challenge in trying to analyze these data on their own. Cloud computing offers access to a large amount of computing resources in an on-demand and pay-per-use fashion, which is a practical way for people to analyze these huge data sets. However, many people are still reluctant to outsource biological sequences to the cloud because they contain sensitive information that should be kept secret for ethical, security, and legal reasons. One of the most fundamental and frequently used computational tools for biological sequence analysis is pairwise sequence alignment (PSA). Previous works for securely solving PSAs at the cloud suffer from poor scalability, i.e., they are unable to exploit the cloud's infrastructure to solve PSAs in parallel because resource-limited users need to be constantly involved in the computations. In this paper, we develop a secure outsourcing algorithm that allows users to solve an arbitrary number of PSAs in parallel at the cloud. Compared with previous works, our algorithm can reduce computing time of a large number of PSAs by more than 50% with as few as 5 computing nodes at the cloud.

1 INTRODUCTION

The exponentially growing amount of biological sequence data holds the potential to rapidly advance scientific knowledge [10, 21]. To efficiently and economically analyze these large-scale data sets, resource-limited users can choose to employ cloud computing. In this computing paradigm, users outsource their computing tasks to a cloud server [19], which contains a large amount of computing resources and offers them on an on-demand and pay-per-use basis [18].

Unfortunately, many people are unwilling to outsource their data to the cloud due to privacy concerns [20]. Specifically, biological sequences are very sensitive and should be kept secret from the cloud for ethical, security, and legal reasons [17]. For example, analyzing plant genetic data at the cloud could disclose an agricultural company's intellectual property. Moreover, since biological sequences often belong to different parties, it is difficult for researchers to perform collaborative analyses at the cloud without compromising the sequence owners' privacy. For instance, to enable collaboratively drug discovery at the cloud, pharmaceutical companies would need to share their sequences, compromising their intellectual property[15]. Therefore, to enable scientists and engineers to collaboratively analyze biological sequences at the cloud, it is important to design secure outsourcing tools that preserve data privacy.

One of the most fundamental mathematical problems in biological sequence analysis is pairwise-sequence alignment (PSA), i.e., finding the least-cost set of edit operations, such as insertion, deletion, and substitution, that transform a sequence into another one. For example, in phylogenetics, scientists compute PSAs to obtain genetic trees that reveal organisms evolutionary relationships; and pharmaceutical companies identify protein functionality for drug discovery by solving many PSAs.

Some existing works on secure outsourcing of large-scale computations may potentially be used to collaboratively solve PSAs at the cloud. In particular, Gennaro et al. [9] propose fully homomorphic encryption (FHE), which allows secure outsourcing of a function to the cloud. Although FHE offers a theoretical privacy guarantee, it requires the user to perform computationally expensive operations and forces the cloud to carry out operations on ciphertexts, which adds significant overhead to already expensive computations at the cloud. Researchers have proposed improvements to the computing time of FHE, e.g., [7], but its overhead remains impractically large for users to analyze large-scale data sets. To avoid the computational complexity introduced by FHE, Wang et al. [22] propose to privately outsource a linear program, which can be used to solve a PSA, by applying a linear algebra-based transformation to the objective and constraints. However, this transformation requires prohibitively expensive matrix multiplications, and results in an optimization problem that prevents users from employing the Simplex algorithm, which is the most efficient solution method for linear programs such as the ones that arise when solving PSAs.

There are some works based on secure multi-party computations (SMC) that allow two users, each holding a private sequence, to compute a PSA by performing local computations and exchanging some data. In the work by Atallah et al. [1], users first protect their privacy with partially homomorphic encryption and then collaborate with each other. Jha et al. [13] further reduce computing times by designing an algorithm where the users first create a custom garbled circuit and then collaboratively evaluate it. Huang et al. [12] achieve further improvements on garbled circuit evaluation times by employing optimizations such as pipelined circuit

ACM-BCB '17, August 20-23, 2017, Boston, MA, USA.

978-1-4503-4722-8/17/08…$15.00
DOI: http://dx.doi.org/10.1145/3107411.3107477

execution, and a reduced number of circuit gates. Although SMC-based approaches allow users to collaboratively solve a PSA within a practical amount of time, their performance depends on the quality of the communication link between the users, and it is unclear how they can be applied to securely solve the computation of a two-user PSA at the cloud.

More importantly, we notice that SMC-approaches share a common serious problem, i.e., users actively participate during the solution of the PSA. The need for users to be online during PSA computations significantly limits the scalability of SMC-approaches, and thus are impractical for the analysis of large-scale biological sequence data sets at the cloud. Specifically, since users only have modest computing resources, they are unable to participate in the solution of more than a few PSAs at a time. Therefore, any practical algorithm for large-scale biological sequence analysis should be parallel and scalable, i.e., the cloud should be able to solve an arbitrary number of PSAs in parallel with minimal computing resources, and without having to continuously interact with the users.

In this paper, we develop a practical and efficient secure outsourcing algorithm that allows two users, each holding a private sequence, to collaboratively solve a large amount of PSAs in parallel at the cloud. Specifically, the users protect their data privacy by concealing its sequences with bilinear maps, which is a cryptographic technique that can be efficiently applied by the users. Then, based on the encrypted sequences, the cloud securely formulates a linear program that is equivalent to the user's PSA. Based on the Simplex algorithm, the cloud solves the linear program, and returns the solution to the users, who efficiently retrieve the optimal solution to the original PSA. To preserve the users' privacy, our algorithm restricts the cloud to operate only on the encrypted sequences and randomized values, rather than on the original PSA's values.

The proposed algorithm requires the users to perform computations with only $O(m)$ and $O(n)$ complexity, which is lower than the complexity of solving a PSA, i.e., $O(mn)$, where m and n are the users' sequence lengths. Furthermore, the proposed algorithm only needs one round of communication between the users and the cloud, which is optimal. We implement the user part of the algorithm on a general purpose PC, and the cloud part on the AWS Elastic Compute Cloud (EC2). Compared to previous works, our algorithm can reduce computing time of a large number of PSAs by more than 50% with as few as 5 computing nodes at the cloud.

2 PROBLEM FORMULATION

System Architecture. We consider an asymmetric three-party system architecture formed by two resource-limited users, i.e., Alice and Bob, and a resource-rich cloud server. We show this architecture in Fig. 1. Alice and Bob each hold a character sequence, i.e., $\mathbf{a} = \{a_1, a_2, \ldots, a_{n-1}\}$ and $\mathbf{b} = \{b_1, b_2, \ldots, b_{m-1}\}$, respectively, and intend to calculate the similarity between their sequences by solving a pairwise sequence alignment (PSA). The elements in $\mathbf{a}$ and $\mathbf{b}$ are taken from a character alphabet $\mathcal{H} = \{h_1, h_2, \ldots, h_p\}$. Let the sequence similarity be defined as the least-cost set of edit operations, such as insertion, deletion, and substitution, needed to transform $\mathbf{x} = \{- \mid \mathbf{a}\}$ into $\mathbf{y} = \{- \mid \mathbf{b}\}$, where $-$ is the null character. Then, the PSA is as follows [8]:

$$\min_{\mathcal{D}} \sum_{[x'_i, y'_j] \in \mathcal{D}} d([x'_i, y'_j]) \tag{1}$$

where the minimum is taken over all edit operation sets $\mathcal{D}$ that transform $\mathbf{x}$ into $\mathbf{y}$, and $[x'_i, y'_j]$ denotes an edit operation based on the characters x'_i and y'_j, and d is the edit operation cost function. If $[x'_i, y'_j]$ denotes a substitution operation, i.e., when $[x'_i, y'_j] = [x_i, y_j]$, then function d outputs a value of matrix $\mathbf{G} \in \mathbb{Z}^{p \times p}$ (e.g., the BLOSUM50 matrix), which stores the substitution cost for pairs of characters in $\mathcal{H}$. In particular, if $[x_i, y_j] = [h_k, h_l]$, then $d([x'_i, y'_j]) = g_{k,l}$, which is the element in the kth row and lth column of $\mathbf{G}$. We denote the substitution costs by $w_{sub}([x'_i, y'_j])$. Further, if $[x'_i, y'_j]$ denotes a deletion or insertion operation, i.e., $[x'_i, y'_j] = [x_i, -]$ or $[x'_i, y'_j] = [-, y_j]$, then d outputs w_{del} or w_{ins}, which are the deletion and insertion costs, respectively.

Due to the large amount of biological sequences, in many applications, Alice and Bob need to collaborate to compute (1) repeatedly many times, and hence, they are unable to solve the PSAs by themselves in a feasible amount of time. For example, in phylogenetic analysis, scientists need to compute a very large amount of PSAs to classify biological sequences, e.g., $\approx 500,000$ PSAs are needed to explore the phylogenetic relationships between viruses within a family [6, 16, 23]. Thus, to solve the PSAs, Alice and Bob need to outsource the most computationally expensive tasks to the cloud.

Threat Model. We consider a semi-honest threat model for all parties in the system, i.e., Alice, Bob, and the cloud. In particular, the cloud attempts to learn Alice's and Bob's private information from their outsourced sequences and from the results of its own computations. Moreover, both Alice and Bob will attempt to extract each others' private information from the results that they receive from the cloud. Therefore, to securely outsource the computation of a PSA problem, the data that Alice and Bob outsource to the cloud should be provably secure [14]. We note that the semi-honest model is a standard threat model for secure computations [11], and it is consistent with previous works in the area of secure multiparty PSA computations, i.e., [12, 13].

Bilinear Map. In our proposed secure PSA outsourcing algorithm, we leverage the properties of pairing-based cryptography (PBC) to conceal Alice and Bob's private sequences. In particular, let $\mathbb{G}_1$ and $\mathbb{G}_2$ be cyclic multiplicative groups of the same prime order q. An admissible bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ is a map with the following property [5]: $e(u^a, v^b) = e(u, v)^{ab}$, for any $u, v \in \mathbb{G}_1$, and $a, b \in \mathbb{Z}_q$, where $\mathbb{Z}_q$ denotes the integers modulo q. The bilinearity property implies that for any $u_1, u_2 \in \mathbb{G}_1$, $e(u_1 \cdot u_2, v) = e(u_1, v) \cdot e(u_2, v)$. Besides, a bilinear map satisfies the non-degeneracy and computability properties. Such an admissible bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ can be implemented by the modified Weil/Tate pairings.

3 PRIVACY-PRESERVING TRANSFORMATIONS

To securely upload a PSA to the cloud, Alice and Bob first secure their private information by performing certain computations. In

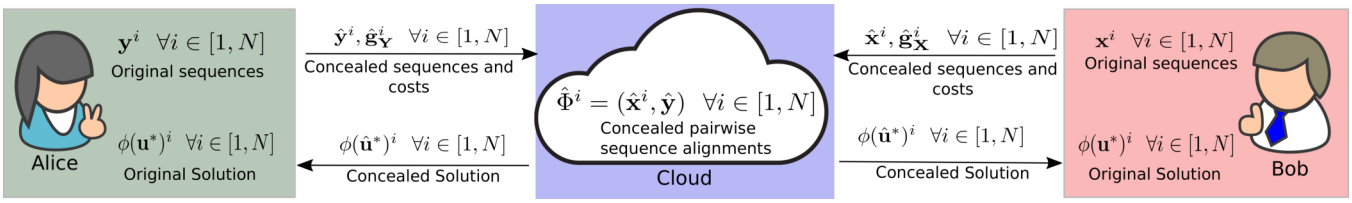


Figure 1: A secure architecture for secure outsourcing of pairwise sequence alignment

this section, we describe an efficient method to conceal a biological sequence and a method to securely obtain a PSA's substitution costs from two concealed sequences. This method can be used by the cloud to securely find the similarity between the sequences.

Privacy-Preserving Sequence Transformation. To conceal its private sequence, a user, e.g., Alice or Bob, first replaces its sequence's characters with integers, and then conceals them through modular multiplications with random numbers. Specifically, let $\mathbf{s} = [s_1, s_2, \ldots, s_m]$ be a private sequence with characters taken from an alphabet $\mathcal{T} = \{t_1, t_2, \ldots, t_p\}$. Then, the user conceals $\mathbf{s}$ as follows:

$$\hat{\mathbf{s}} = \mathbf{V}_{\mathbf{s}}\bar{\mathbf{s}}. \tag{2}$$

The elements of vector $\bar{\mathbf{s}} = [\mathcal{M}(s_1), \mathcal{M}(s_2), \ldots, \mathcal{M}(s_m)]$ are given by a function $\mathcal{M} : \mathcal{T} \rightarrow \mathbb{G}_1$ that uniquely maps characters in $\mathcal{T}$ to integers in the multiplicative cyclic group $\mathbb{G}_1$ of prime order q. The matrix $\mathbf{V}_{\mathbf{s}} \in \mathbb{G}_1^{m \times m}$ is diagonal with non-zero elements given by the vector $\mathbf{v}_{\mathbf{s}} = [h^{\sigma_1}, \ldots, h^{\sigma_m}]$, where h is a generator of $\mathbb{G}_1$, and σ_i is a randomly chosen element from the set of integers less than q, which we denote by $\mathbb{Z}_q$ (for all $i \in [1, m]$).

Consequently, the elements of $\hat{\mathbf{s}}$ in (2) are given by $\hat{s}_i = \bar{s}_i h^{\sigma_i} \mod q$ (for all $i \in [1, m]$), where q is a large prime number. Besides, the user calculates a vector of concealed random numbers $\hat{\mathbf{v}}_{\mathbf{s}} = [e(h, h)^{\nu_1}, \ldots, e(h, h)^{\nu_m}]$, where $\nu_i = q - \sigma_i \mod q$, and $e(\cdot, \cdot)$ is the bilinear map defined in Section 2.

Privacy-Preserving Substitution Cost Transformation. To allow the cloud to securely formulate the substitution costs in the PSA based on the concealed sequences, Alice and Bob need to upload a concealed version of the substitution costs to the cloud.

Specifically, to find integers that represent all possible character pairs in alphabet $\mathcal{T}$, Alice builds a matrix $\mathbf{Z} \in \mathbb{G}_2^{p \times p}$ with elements given by $z_{i,j} = e(\bar{t}_i \cdot \bar{t}_j, h)$ (for all $i, j \in [1, p]$), where h is a generator of $\mathbb{G}_1$, $\bar{t}_i$ and $\bar{t}_j$ are the integer representations of t_i and t_j given by $\mathcal{M}$ as described in (2), respectively, and e is the bilinear map described in Section 2. Besides, we denote a function that maps elements in $\mathbf{Z}$ to substitution costs in $\mathbf{G}$, i.e., $\psi(z_{i,j}) = g_{i,j} = w_{sub}([t_i, t_j])$ (for all $i, j \in [1, p]$).

Alice then forms buckets of elements $z_{i,j}$'s that are mapped by ψ to the same substitution cost in $\mathbf{G}$, that is, $\mathcal{B}_l = \{z_{i,j} \mid \psi(z_{i,j}) = g'_l\}$ for all $l \in [1, p']$, where g'_l is the lth unique substitution cost in $\mathbf{G}$, and p' is the total number of unique substitution costs. We denote a second function $\tilde{\psi}$ that maps $z_{i,j}$ to its corresponding bucket's index l.

Now, after Alice uploads the concealed sequence $\hat{\mathbf{s}} \in \mathbb{G}_1^{m \times 1}$ to the cloud as described in Section 3, she uploads the set of concealed costs for every element s_i which is given by

$$\hat{\mathbf{g}}_{s_i} = \beta^{\frac{1}{2}} \gamma_{s_i} \mathbf{g}'^{\frac{1}{2}} \tag{3}$$

for all $i \in [1, m]$, where $\mathbf{g}'$ is the vector of unique substitution costs in $\mathbf{G}$, and β and γ_{s_i} are random positive integers in $\mathbb{Z}$. Alice uploads $\hat{\mathbf{g}}_{s_i}$ (for all $i \in [1, m]$) to the cloud.

Similarly, Bob can follow the same procedure to find its concealed sequence, and upload his concealed substitution costs, i.e,. $\hat{\mathbf{g}}_{f_i} = \beta^{\frac{1}{2}} \gamma_{f_i} \mathbf{g}'^{\frac{1}{2}}$, where $\mathbf{f}$ is Bob's private sequence. In Section 5, we explain how the cloud can employ the outsourced sequences, and the results of (3) to find the PSA's concealed substitution costs.

Note that the size of vectors $\hat{\mathbf{g}}_{s_i}$ is determined by the number of unique costs p'. In practice, this quantity is small, e.g., 18 for the BLOSUM50 matrix, and thus it only adds a small constant overhead to the user's memory and communication complexities.

4 A LINEAR PROGRAM FOR PAIRWISE SEQUENCE ALIGNMENT

In this section, we reformulate the PSA in (1) as a linear program (LP), i.e., an optimization problem with a linear objective, and affine constraints, that can be efficiently solved by the cloud using the Simplex algorithm. We also show that the solution to the PSA is readily available from the solution of the LP.

We first express (1) as a shortest-path problem on a directed acyclic graph (DAG) $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where the node set $\mathcal{V}$ and edge set $\mathcal{E}$ are based on the cost of edit operations between sequences $\mathbf{x}$ and $\mathbf{y}$. Specifically, we first store the cost of substituting elements in $\mathbf{x}$ for elements in $\mathbf{y}$ in a matrix $\mathbf{T} \in \mathbb{Z}^{m \times n}$ as follows:

$$t_{i,j} = w_{sub}([x_i, y_j]) \tag{4}$$

for all $i \in [1, m]$ and $j \in [1, n]$, where $w_{sub}(\cdot)$ is as defined in Section 2.

We then form the node set $\mathcal{V}$ by adding a node for each element in $\mathbf{T}$, that is, $\mathcal{V} = \{v_p \mid p = i + m(j-1)\ \forall i \in [1, m],\ \forall j \in [1, n]\}$, where the node v_p corresponds to the element $t_{i,j}$. To form the edge set $\mathcal{E}$, we arrange $\mathcal{V}$'s nodes on a two-dimensional grid according to their positions in $\mathbf{T}$, and add a directed edge from each node to its bottom, right, and bottom-right neighbors, that is, $\mathcal{E} = \mathcal{E}_{del} \cup \mathcal{E}_{ins} \cup \mathcal{E}_{sub}$, where $\mathcal{E}_{del} = \{(v_p, v_q) \mid q = p+1\}$, $\mathcal{E}_{ins} = \{(v_p, v_q) \mid q = p+m\}$, $\mathcal{E}_{sub} = \{(v_p, v_q) \mid q = p+m+1\}$, and (v_p, v_q) is an ordered pair denoting a directed edge from start node v_p to end node v_q.

The weights for edges corresponding to substitutions are given by the elements of $\mathbf{T}$, i.e., $w_l = t_{\pi(v_l)}$, if $e_l \in \mathcal{E}_{sub}$, where e_l is the lth element of $\mathcal{E}$, and v_l is e_l's end node. The function $\pi(v_l)$ returns v_l's two-dimensional index in $\mathbf{T}$, i.e., $\pi(v_l) = (i, j)$, where $i = (l-1 \mod m) + 1$, and $j = \lceil l/m \rceil$. Weights for edges corresponding to deletion and insertion operations are given by w_{del}, if $e_l \in \mathcal{E}_{del}$ and by w_{ins}, if $e_l \in \mathcal{E}_{ins}$, where w_{del} and w_{ins} are the deletion and insertion costs defined in Section 2, respectively.

Consequently, the shortest-path between v_1 and v_{mn} on $\mathcal{G}$ can be defined by an integer LP as follows:

$$\min_{\mathbf{z}} \phi(\mathbf{z}) = \mathbf{w}^T\mathbf{z} \text{ subject to } \mathbf{Az} = \mathbf{b},\ \mathbf{z} \geq \mathbf{0},\ \mathbf{z} \in \mathbb{Z}, \quad (5)$$

where $\mathbf{z} \in \mathbb{Z}^{|\mathcal{E}|}$ is the optimization variable. We form the node-edge incidence matrix $\mathbf{A} \in \mathbb{Z}^{|\mathcal{V}|\times|\mathcal{E}|}$ by placing $a_{i,l} = 1$ if node i is a start node of edge e_l, and $a_{i,l} = -1$, if node i is the end node of e_l. The rest of the elements are set to zero. We set the element that corresponds to the source node in the vector $\mathbf{b} \in \mathbb{Z}^{|\mathcal{V}|}$ equal to 1, i.e., $b_1 = 1$, and the element that corresponds to the destination node equal to -1, that is, $b_{mn} = -1$. We denote the solution of (5) by $\mathbf{z}^*$. Note that due to the particular definitions of $\mathbf{A}, \mathbf{b}$, and $\mathbf{c}$, the elements of $\mathbf{z}$ can only take binary integer values, i.e,. $\{0, 1\}$, even though $\mathbf{z}$ is constrained as a non-negative integer in (5).

Since the shortest-path between v_1 and v_{mn} on $\mathcal{G}$ can be interpreted as the least-cost set of edit operations that transforms $\mathbf{x}$ into $\mathbf{y}$, the solution to (1) based on $\mathbf{z}^*$ is given by: $\mathcal{D}^* = \{[x'_i, y'_j] \mid z^*_l > 0 \wedge \pi(v_l) = (i, j)\}$, where $[x'_i, y'_j]$ has a one-to-one mapping to a deletion, insertion, or substitution operation according to whether edge e_l belongs to $\mathcal{E}_{del}, \mathcal{E}_{ins}$ or $\mathcal{E}_{sub}$, respectively.

A Relaxed Optimization Problem for PSA. In general, integer LPs such as (5) are very difficult to solve because they are NP-hard. However, since (5) is defined by integers, i.e., $\mathbf{A}$, $\mathbf{b}$, and $\mathbf{c}$ all have integer values, we can relax (5) to form a continuous LP that can be efficiently solved by the Simplex algorithm in polynomial time, and yields the solution to (5). Specifically, by removing the integrality constraint in (5), we get the following continuous LP, i.e.,

$$\min_{\mathbf{u}} \phi(\mathbf{u}) = \mathbf{w}^T\mathbf{u} \text{ subject to } \mathbf{Au} = \mathbf{b},\ \mathbf{u} \geq \mathbf{0}, \quad (6)$$

where $\mathbf{u}$ is the optimization variable. We denote the optimal solution of the relaxed problem by $\mathbf{u}^*$.

Since the values in $\mathbf{A}$, $\mathbf{w}$ and $\mathbf{b}$ in the relaxed problem in (6) are all integers, the Simplex algorithm's computations result in integer values, including the solution $\mathbf{u}^*$ [3]. Therefore, the optimal solution of the LP in (6) found by the Simplex algorithm, is also the optimal solution of the integer LP in (5), and ultimately to the PSA in (1). In the following, we describe how the cloud can use the Simplex algorithm to solve a secure version of (6).

5 SECURE OUTSOURCING OF PAIRWISE SEQUENCE ALIGNMENT

In this section, we describe our secure PSA solver (SPSA), which efficiently and securely solves the PSA in (1) in parallel at the cloud. Specifically, Alice and Bob conceal their private sequences and substitution cost matrix with the transformations in Section 3, and collaborate with the cloud to securely formulate the LP in (6). Then, by applying the Simplex algorithm, the cloud solves the secure LP and returns the results to Alice and Bob, who efficiently find the solution to the original PSA.

Secure Computation of the Similarity Matrix. To securely outsource the computation of the similarity matrix $\mathbf{T}$ in Section 4, Alice and Bob conceal sequences $\mathbf{y}$ and $\mathbf{x}$ in (1), respectively, compute the secure substitution costs, and then the cloud employs these outsourced data to securely find the edit operation costs. In particular, Alice and Bob conceal their sequences as follows: $\hat{\mathbf{y}} = \mathbf{V_y}\bar{\mathbf{y}}$, $\hat{\mathbf{x}} = \mathbf{V_x}\bar{\mathbf{x}}$, where $\mathbf{V_y}$ and $\mathbf{V_x}$ are computed as in (2), and $\bar{\mathbf{y}} \in \mathbb{G}_1^{m\times 1}$, and $\bar{\mathbf{x}} \in \mathbb{G}_1^{n\times 1}$ are the integer representations of $\mathbf{x}$ and $\mathbf{y}$ as described in Section 3, respectively. In this step, Alice and Bob also find the vectors of concealed random numbers $\hat{\mathbf{v}}_\mathbf{y}$ and $\hat{\mathbf{v}}_\mathbf{x}$.

Then, Alice forms the buckets $\mathcal{B}_i$ for every unique substitution cost as in Section 3, and securely shares them with Bob using public key cryptography techniques, e.g., RSA cryptosystem. Similarly, Alice and Bob generate the random numbers α and β, respectively, and securely share them with each other.

Next, Alice and Bob generate their concealed cost vectors for $\mathbf{x}$ and $\mathbf{y}$, respectively, that is, $\hat{\mathbf{g}}_{y_i} = \beta^{\frac{1}{2}}\gamma_{y_i}\mathbf{g'}^{\frac{1}{2}}$, $\hat{\mathbf{g}}_{x_j} = \beta^{\frac{1}{2}}\gamma_{x_j}\mathbf{g'}^{\frac{1}{2}}$ (for all $i \in [1, m]$ and $j \in [1, n]$), where γ_{y_i} and γ_{x_j} are computed as in (3). We denote the vectors of random numbers used to conceal the substitution costs by $\boldsymbol{\gamma}_\mathbf{y}$ and $\boldsymbol{\gamma}_\mathbf{x}$, respectively. Alice and Bob keep $\boldsymbol{\gamma}_\mathbf{y}$, $\boldsymbol{\gamma}_\mathbf{x}$, β, and α secret.

Alice also conceals the insertion and deletion operation costs as follows: $\hat{w}_{ins} = \beta\gamma_{ins}w_{ins}$, $\hat{w}_{del} = \beta\gamma_{del}w_{del}$, where $\gamma_{ins} \in \mathbb{Z}$ and $\gamma_{del} \in \mathbb{Z}$ are random positive numbers. Besides, both Alice and Bob conceal their respective random numbers, i.e., $\hat{\boldsymbol{\gamma}}_\mathbf{y} = \alpha^{\frac{1}{2}}\boldsymbol{\gamma}_\mathbf{y}$, $\hat{\boldsymbol{\gamma}}_\mathbf{x} = \alpha^{\frac{1}{2}}\boldsymbol{\gamma}_\mathbf{x}$, $\hat{\gamma}_\alpha = \alpha\gamma_\alpha$, $\hat{\gamma}_{ins} = \alpha\gamma_{ins}$, $\hat{\gamma}_{del} = \alpha\gamma_{del}$. Once Alice completes her computations, she uploads $\hat{\mathbf{y}}$, $\hat{\mathbf{g}}_{y_i}$ (for all $i \in [1, m]$), $\hat{w}_{ins}, \hat{w}_{del}$, $\hat{\boldsymbol{\gamma}}_\mathbf{y}$, $\hat{\gamma}_\alpha$,$\hat{\gamma}_{ins} = \alpha\gamma_{ins}$, $\hat{\gamma}_{del} = \alpha\gamma_{del}$, and the buckets $\mathcal{B}_l$ (for all $l \in [1, p']$). Similarly, Bob uploads $\hat{\mathbf{x}}$, $\hat{\mathbf{g}}_{x_j}$ (for all $j \in [1, n]$), and $\hat{\boldsymbol{\gamma}}_\mathbf{x}$.

After receiving the concealed values, the cloud securely finds the elements of the similarity matrix in (4) by first computing the product of bilinear mappings between characters in $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$, and then mapping them to their corresponding buckets. In particular, the cloud first forms the matrix $\bar{\mathbf{T}} \in \mathbb{G}_2^{m\times n}$ with values defined by: $\bar{t}_{i,j} = e(\hat{y}_i\hat{x}_j, h) \cdot \hat{v}_{y_i} \cdot \hat{v}_{x_j}$. We show the correctness of $\bar{t}_{i,j}$ as follows:

$$\begin{aligned}\bar{t}_{i,j} &= e(\hat{y}_i\hat{x}_j, h) \cdot \hat{v}_{y_i} \cdot \hat{v}_{x_j} \\ &= e(\bar{y}_i h^{\sigma_{y_i}} \cdot \bar{x}_j h^{\sigma_{x_j}}, h) \cdot e(h, h)^{\nu_{y_i}} \cdot e(h, h)^{\nu_{x_j}} \\ &= e(\bar{y}_i\bar{x}_j, h) \cdot e(h^{\sigma_{y_i}+\sigma_{x_j}}, h) \cdot e(h, h)^{\nu_{y_i}+\nu_{x_j}} \\ &= e(\bar{y}_i\bar{x}_j, h).\end{aligned}$$

where $\nu_{y_i} = q - \sigma_{y_i} \mod q$, and $\nu_{x_j} = q - \sigma_{x_j} \mod q$.

Based on the mapping $\tilde{\psi}$ between elements in $\bar{\mathbf{T}}$ and bucket indexes, the cloud computes the secure substitution cost matrix $\hat{\mathbf{T}} \in \mathbb{Z}^{m\times n}$ as follows: $\hat{t}_{i,j} = \hat{g}'_{y_i,l}\hat{g}'_{x_j,l} = (\beta\gamma_{y_j}\gamma_{x_j})g_l$, (for all $i \in [1, m]$ and $j \in [1, n]$), where l is the index of bucket $\mathcal{B}_l$ that contains $\bar{t}_{i,j}$, and $\hat{g}'_{y_i,l}$ and $\hat{g}'_{x_j,l}$ are the lth elements of $\hat{\mathbf{g}}'_{x_i}$ and $\hat{\mathbf{g}}'_{y_j}$, respectively. Due to the properties of the bilinear map and the finite groups used to conceal the private sequences, the cloud is unable to compute $\mathbf{x}$ and $\mathbf{y}$ based on $\bar{\mathbf{T}}$, $\hat{\mathbf{T}}$, $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$.

Secure Formulation of the Linear Program. To securely formulate the LP described in (6), the cloud needs to form the secure weight vector $\hat{\mathbf{w}} \in \mathbb{Z}^{|\mathcal{E}|}$ and the secure constraint matrix $\hat{\mathbf{A}} \in \mathbb{Z}_*^{|\mathcal{V}|\times|\mathcal{E}|}$. To this end, the cloud first forms the secure weight vector $\hat{\mathbf{w}}$ by assigning elements of $\hat{\mathbf{T}}$ to substitution edges, i.e., $w_l = \hat{t}_{\pi(l)}$, if $e_l \in \mathcal{E}_{sub}$, and by assigning the secure insertions and deletion costs to the corresponding edges, i.e., $\hat{w}_l = \hat{w}_{del}$, if $e_l \in \mathcal{E}_{del}$ and $\hat{w}_l = \hat{w}_{ins}$, if $e_l \in \mathcal{E}_{ins}$.

To form $\hat{\mathbf{A}}$, the cloud computes the following matrix of concealed random numbers: $\hat{\mathbf{P}} = (\hat{\boldsymbol{\gamma}}_{\mathbf{y}}^{\top}\hat{\boldsymbol{\gamma}}_{\mathbf{x}})$ and then forms $\hat{\mathbf{A}}'$'s columns vectors by multiplying columns corresponding to substitution edges by elements in $\hat{\mathbf{P}}$, that is, $\hat{\mathbf{a}}'_j = \hat{p}_{\pi(j)}\mathbf{a}_j$ if $e_j \in \mathcal{E}_{sub}$, and columns corresponding to insertion and deletion edges by the concealed weights, that is, $\hat{\mathbf{a}}'_j = \hat{\gamma}_{del}\mathbf{a}_j$, if $e_j \in \mathcal{E}_{del}$, $\hat{\mathbf{a}}'_j = \hat{\gamma}_{ins}\mathbf{a}_j$, if $e_j \in \mathcal{E}_{ins}$, where $\mathbf{a}_j$ is the jth column of $\mathbf{A}$ in (6), $\hat{p}_{\pi(j)}$ is the element in the ith row and jth column of $\hat{\mathbf{P}}$ as defined by $\pi(\cdot)$ in Section 4. Besides, the cloud multiplies the rows of $\hat{\mathbf{A}}'$ by $\hat{\gamma}_{\alpha}$, that is $\hat{\mathbf{A}} = \mathbf{K}\hat{\mathbf{A}}'$ where $\mathbf{K} = \hat{\gamma}_{\alpha}\mathbf{I}$ and $\mathbf{I} \in \mathbb{Z}_*^{|\mathcal{V}|\times|\mathcal{E}|}$ is the identity matrix.

To compute the concealed constraint vector, the cloud computes the following: $\hat{\mathbf{b}} = \hat{\gamma}_{\alpha}\mathbf{b}$, where $\mathbf{b}$ is the constant vector in (6). Note that the cloud can compute $\mathbf{A}$ and $\mathbf{b}$ based on the LP formulation in Section 4 and the length of sequences $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ that Alice and Bob upload.

Secure solution of the Linear Program. Before the cloud can apply the Simplex algorithm to solve the secure LP, it needs to find an initial solution. Specifically, the Simplex algorithm requires a basic feasible starting point $\mathbf{u}$ such that a matrix $\mathbf{B} \in \mathbb{Z}^{|\mathcal{V}|\times|\mathcal{V}|}$ is invertible, and $\mathbf{u}^{\mathbf{B}} > \mathbf{0}$ and $\mathbf{u}^{\mathbf{N}} = \mathbf{0}$. Matrix $\mathbf{B}$ is formed with $|\mathcal{V}|$ columns of $\mathbf{A}$, and $\mathbf{u}^{\mathbf{B}} \in \mathbb{Z}^{|\mathcal{V}|\times 1}$ with the elements of $\mathbf{u}$ that correspond to these columns. Vector $\mathbf{u}^{\mathbf{N}} \in \mathbb{Z}^{(|\mathcal{E}|-|\mathcal{V}|)\times 1}$ is formed with the remaining values of $\mathbf{u}$. Since we are dealing with a shortest-path problem on a DAG, we can form a feasible solution for (6) based on any path between the source node and the sink node. However, since such a path contains less than $|\mathcal{V}|$ edges, its corresponding initial solution is not basic. To obtain an initial basic feasible solution, we find an arbitrary path between all nodes and the destination node, and redefine the constraint vector $\mathbf{b}$ in (6) as $b_i = 1$ (for all $i \in [1, mn-1]$) and $b_{mn} = \sum_{i=1}^{mn-1} -b_i$. Note that the solution to our original problem is still readily available from the solution to the LP with this choice of $\mathbf{b}$.

The cloud can find the initial basic feasible solution for the secure LP by computing the following: $\hat{\mathbf{u}} = \hat{\mathbf{B}}^{-1}\hat{\mathbf{b}}$, where $\hat{\mathbf{B}}$ is the set of basic feasible columns of $\hat{\mathbf{A}}$ based on a path between the source and destination nodes.

After the cloud finds the concealed starting basic feasible solution $\hat{\mathbf{u}}$, it executes the revised Simplex algorithm's main iteration with the secure LP's matrices until convergence [3]. Once the cloud finds the solution to the secure LP, i.e., $\hat{\mathbf{u}}^*$, it calculates a concealed optimal objective value as follows: $\phi(\hat{\mathbf{u}}^*) = \hat{\mathbf{w}}^{\top}\hat{\mathbf{u}}^*$, and transmits it to the users. Finally, both Alice and Bob can recover the original objective value by computing $\phi(\mathbf{u}^*) = \alpha\beta^{-1}\phi(\hat{\mathbf{u}}^*)$. Note that since we are dealing with a shortest path problem on a DAG, the revised Simplex algorithm is guaranteed to converge.

Since Alice and Bob only perform operations with their own sequences, their overall computational complexity in the SPSA algorithm is $O(n)$ and $O(m)$, respectively, which is lower than $O(mn)$, i.e., the complexity for users solving PSAs by themselves.

Also note that the cloud is unable to learn private information about the users because it only has access to the securely transformed sequences, and the securely transformed edit operation costs. In particular, the cloud would need to find the σ_{y_i}'s and σ_{x_i}'s used to transform $\mathbf{x}$ and $\mathbf{y}$ based on the elements of $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$. However, finding σ_{y_i}'s or σ_{x_i}'s requires the cloud to break the PBC encryption in Section 2 by solving a discrete logarithm under a multiplicative cyclic group $\mathbb{G}_1$ of order q, which has no known polynomial time solution.

A Parallel Implementation for the SPSA. Since Alice and Bob are interested in solving (1) repeatedly many times as described in Section 2, we propose an scheme for them to solve a large amount of PSAs at the cloud in parallel. Specifically, suppose Alice and Bob hold the private sequence sets $\mathbf{Y} = \{\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^N\}$ and $\mathbf{X} = \{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^N\}$, respectively, and outsource to the cloud the following PSAs: $\hat{\Phi}^i = (\hat{\mathbf{y}}^i, \hat{\mathbf{x}}^i)$ (for all $i \in [1, N]$), where $(\hat{\mathbf{y}}^i, \hat{\mathbf{x}}^i)$ is a tuple that denotes the alignment problem in (1) with the concealed sequences as inputs, and N is the number of sequence pairs held by Alice and Bob.

Since the sequence transformation and concealed PSAs $\hat{\Phi}^i$'s are independent from each other, they can be computed in parallel. In particular, to conceal their sequences, Alice first finds the buckets $\mathcal{B}_l^i$'s, and random numbers α^i as described in Section (3) for all $i \in [1, N]$, and securely shares them with Bob. Similarly, Bob finds β^i for all $i \in [1, N]$ and securely sends them to Alice. Then, Alice and Bob compute $\hat{\mathbf{Y}}$ and $\hat{\mathbf{X}}$, i.e., the concealed sequences in $\mathbf{Y}$ and $\mathbf{X}$, and the concealed substitution costs $\hat{\mathbf{g}}$'s for all characters and for all sequences, and upload them to the cloud. After the cloud receives the concealed values, it can solve $\hat{\Phi}^i$ for all $i \in [1, N]$ in parallel, without interacting with neither Alice or Bob. After solving the PSAs, the cloud returns the results to Alice and Bob, who then retrieve the solution to the original PSA by computing: $\phi(\mathbf{u}^{*(i)}) = \alpha^i\beta^{-1(i)}\phi(\hat{\mathbf{u}}^{*(i)})$. Note that finding the buckets $\mathcal{B}_l$'s and generating random scalars α^i and β^i for a large number of PSAs are one-time operations and can be done efficiently. Besides, the cloud can begin to solve the PSAs as soon as a pair of concealed sequences becomes available, without the need to wait for Alice and Bob to finish concealing their sequences.

6 EXPERIMENT RESULTS

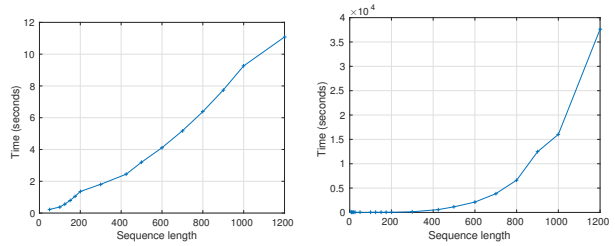
To closely replicate a practical cloud computing scenario, we run Alice and Bob's part of the SPSA algorithm on a laptop with a dual-core 2.6GHz CPU, 8GB RAM memory, and a 150GB solid state drive, and the cloud part on an Amazon Elastic Compute Cloud (EC2) cluster using m4.large instances, which each have 2 CPUs and 8GB RAM. We implement the linear algebra computations on Matlab 2015b, the pairing-based cryptography operations in C using the Pairing-Based Cryptography (PBC) Library, and the Simplex algorithm with IBM's CPLEX. We test the performance of our algorithm with beta protein sequences taken from the Protein Data Bank [2] of the same size, that is $n = m$. We also employ randomly generated sequences to compare the SPSA with previous works.

We first analyze the computing time of our secure outsourcing algorithm SPSA at both the user and at the cloud. In Fig. 2(a), we observe that a user, i.e., Alice or Bob, can transform sequences very quickly, even for large n and m. For example, the computing time of the user when the length of its sequence is 1000 characters is only 9.3s. Besides, in Fig. 2(b), we observe that the computing time of the cloud increases with the length of the sequences.

We also investigate the scalability of our SPSA algorithm by comparing its overall computing time with different cloud cluster sizes with the least computing time that we can expect. To find this theoretical lower bound on the computing time of the cloud, we take the overall computing time with a one-node cluster and

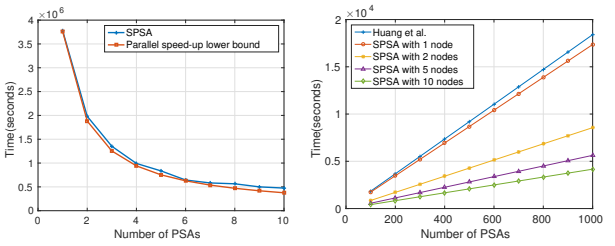
divide it by the number of nodes in the cluster. Fig. 2(c) shows both the SPSA's computing time and the lower bound on the computing time for solving 100 PSAs under different cluster sizes. We observe that our algorithm is very close to the lower bound on the computing time. This is due to the ability of the SPSA to independently solve linear programs at different nodes. Therefore, our secure outsourcing algorithm is scalable, which is a crucial requirement to solve a large number of PSAs within a feasible amount of time.

Next, we compare the overall computing time of the SPSA algorithm with different cluster sizes with that of [12] in Fig. 2(d). We observe that our algorithm offers significant time savings compared to [12] with only a modest amount of computing nodes at the cloud. For example, in Fig. 2(d), we observe that for 500 PSAs with input sequences of length $n = m = 200$ the algorithm by [12] takes 183 minutes, while the SPSA algorithm with a 5-node cluster only takes 83 minutes. This is due to the ability of the SPSA to solve many linear programs at the cloud in parallel without having to interact with the users. In contrast, the garbled circuits in [12] require the users to participate in circuit evaluation, which prevents them from solving PSAs in parallel[1].



(a) Computing time at the users for problems with sequences of different sizes with $n = m$.

(b) Computing time at the cloud for problems with sequences of different sizes with $n = m$.



(c) Overall computing time for different number of nodes at the cloud for 100 PSAs with sequence sizes $n = m = 1200$.

(d) Performance comparison of our SPSA with [4] for sequences of size $m = n = 1000$ with different cluster sizes.

Figure 2: Performance evaluation of the proposed secure pairwise sequence alignment (SPSA) algorithm.

7 CONCLUSIONS

In this paper, we have investigated the problem of two users, each holding a private sequence, securely outsourcing pairwise sequence alignments (PSAs) to the cloud. To protect the users' private data, we develop a sequence transformation scheme that has low computational complexity. Then, based on the Simplex algorithm, the cloud solves the PSA under the transformed sequences, and sends the results to the users, who can find the solution to their original PSA with minimal computing resources. Different from previous works, our algorithm can solve an arbitrary number of PSAs at the cloud in parallel, which significantly reduces the computing time. Our results show that the proposed algorithm is scalable and can reduce computing time of a large number of PSAs by more than 50% with as few as 5 computing nodes at the cloud.

[1]The authors in [12] implemented their algorithm on a PC with similar characteristics to the EC2 m4.large instance used in our simulations

REFERENCES

[1] M. J. Atallah, F. Kerschbaum, and W. Du. Secure and private sequence comparisons. In *ACM Workshop on Privacy in the Electronic Society*, WPES '03, Washington, DC, 2003.

[2] H. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, H. Weissig, I. Shindyalov, and P. Bourne. The protein data bank. *Nucleic Acids Research*, 28:235–242, 2000.

[3] D. Bertsimas and J. N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific-Dynamic Ideas, Nashua, NH, USA, 1997.

[4] M. Blanton, M. J. Atallah, K. B. Frikken, and Q. Malluhi. Secure and efficient outsourcing of sequence comparisons. In *European Symposium on Research in Computer Security (ESORICS)*, Pisa, Italy, September 2012.

[5] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the weil pairing. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 514–532. Springer, 2001.

[6] J. K. Brown, F. M. Zerbini, J. Navas-Castillo, E. Moriones, R. Ramos-Sobrinho, J. C. Silva, E. Fiallo-Olivé, R. W. Briddon, C. Hernández-Zepeda, A. Idris, et al. Revision of begomovirus taxonomy based on pairwise sequence comparisons. *Archives of virology*, 160(6):1593–1619, 2015.

[7] C. Costello, C. Fournet, J. Howell, M. Kohlweiss, B. Kreuter, M. Naehrig, B. Parno, and S. Zahur. Geppetto: Versatile verifiable computation. In *IEEE Symposium on Security and Privacy*, pages 253–270. IEEE, 2015.

[8] R. Durbin, S. Eddy, and A. krogh anf G. Mitchison. *Biological Sequence Anlaysis*. Cambridge University Press, Cambridge, UK, 1998.

[9] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology (CRYPTO)*, Santa Barbara, CA, USA, 2010.

[10] D. Gevers, R. Knight, J. F. Petrosino, K. Huang, A. L. McGuire, B. W. Birren, K. E. Nelson, O. White, B. A. Meth, and C. Huttenhower. The human microbiome project: A community resource for the healthy human microbiome. *PLoS Biol*, 10(8):1–5, 08 2012.

[11] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 2004.

[12] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *USENIX Conference on Security*, San Francisco, CA, USA, August 2011.

[13] S. Jha, L. Kruger, and V. Shmatikov. Towards practical privacy for genomic computation. In *IEEE Symposium on Security and Privacy*, Oakland, CA, USA, 2008.

[14] J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC, 2008.

[15] Organisation for Economic Co-operation and Development (OECD). Collaborative mechanisms for intellectual property management in the life sciences. http://www.oecd.org/sti/biotech/48665248.pdf, 2011.

[16] B. M. Muhire, A. Varsani, and D. P. Martin. Sdt: A virus classification tool based on pairwise sequence alignment and identity calculation. *PLOS ONE*, 9(9):1–8, 09 2014.

[17] H. K. Patil and R. Seshadri. Big data security and privacy issues in healthcare. In *IEEE International Congress on Big Data*, Anchorage, AK, USA, June 2014.

[18] S. Sakr, A. Liu, D. Batista, and M. Alomari. A survey of large scale data management approaches in cloud environments. *IEEE Communications Surveys Tutorials*, 13(3):311–336, March 2011.

[19] Y. Simmhan, S. Aman, A. Kumbhare, R. Liu, S. Stevens, Q. Zhou, and V. Prasanna. Cloud-based software platform for big data analytics in smart grids. *Computing in Science Engineering*, 15(4):38–47, July 2013.

[20] F. Stajano, L. Bianchi, P. Liò, and D. Korff. Forensic genomics: Kin privacy, driftnets and other open questions. In *ACM Workshop on Privacy in the Electronic Society*, Alexandria, Virginia, USA, 2008.

[21] Z. D. Stephens, S. Y. Lee, F. Faghri, R. H. Campbell, C. Zhai, M. J. Efron, R. Iyer, M. C. Schatz, S. Sinha, and G. E. Robinson. Big data: Astronomical or genomical? *PLoS Biol*, 13(7):1–11, July 2015.

[22] C. Wang, K. Ren, and J. Wang. Secure and practical outsourcing of linear programming in cloud computing. In *International Conference on Computer Communications*, Shangai, China, 2011.

[23] X. Xia. Phypa: Phylogenetic method with pairwise sequence alignment outperforms likelihood methods in phylogenetics involving highly diverged sequences. *Molecular Phylogenetics and Evolution*, 102:331 – 343, 2016.