1-1-2018

Secure Similar Sequence Query on Outsourced Genomic Data

Ke Cheng
Boise State University

Yantian Hou
Boise State University

Liangmin Wang
Jiangsu University

Secure Similar Sequence Query on Outsourced Genomic Data

Ke Cheng*
Boise State University
Boise, Idaho, USA
kecheng@boisestate.edu

Yantian Hou
Boise State University
Boise, Idaho, USA
yantianhou@boisestate.edu

Liangmin Wang
Jiangsu University
Zhenjiang, Jiangsu, China
wanglm@ujs.edu.cn

ABSTRACT

The growing availability of genomic data is unlocking research potentials on genomic-data analysis. It is of great importance to outsource the genomic-analysis tasks onto clouds to leverage their powerful computational resources over the large-scale genomic sequences. However, the remote placement of the data raises personal-privacy concerns, and it is challenging to evaluate data-analysis functions on outsourced genomic data securely and efficiently. In this work, we study the secure similar-sequence-query (SSQ) problem over outsourced genomic data, which has not been fully investigated. To address the challenges of security and efficiency, we propose two protocols in the *mixed form*, which combine two-party secure secret sharing, garbled circuit, and partial homomorphic encryptions together and use them to jointly fulfill the secure SSQ function. In addition, our protocols support multi-user queries over a joint genomic data set collected from multiple data owners, making our solution scalable. We formally prove the security of protocols under the semi-honest adversary model, and theoretically analyze the performance. We use extensive experiments over real-world dataset on a commercial cloud platform to validate the efficacy of our proposed solution, and demonstrate the performance improvements compared with state-of-the-art works.

CCS CONCEPTS

• **Security and privacy → Privacy-preserving protocols**; **Security protocols**; **Privacy protections**;

KEYWORDS

Secure similar sequence query, genomic data outsourcing, mixed protocols

ACM Reference Format:
Ke Cheng*, Yantian Hou, and Liangmin Wang. 2018. Secure Similar Sequence Query on Outsourced Genomic Data. In *ASIA CCS'18: 2018 ACM Asia Conference on Computer and Communications Security, June 4–8, 2018, Incheon, Republic of Korea.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3196494.3196535

*Ke Cheng is also a member of Department of Computer Science at Anhui University.

ASIA CCS'18, June 4–8, 2018, Incheon, Republic of Korea

ACM ISBN 978-1-4503-5576-6/18/06…$15.00
https://doi.org/10.1145/3196494.3196535

1 INTRODUCTION

The rapid advance of the gene-sequencing technique is generating a huge amount of genomic data which holds the key to the understanding of many diseases. The cost of sequencing a human genome is about $1,000 today and is estimated to drop to only $100 in near future with the aid of next-generation sequencing techniques [1]. The growing availability of genomic data is unlocking research potentials on genome-data query and analysis [21] in medical domain. Similar Sequence Query (SSQ) on genomic data has been utilized as a cornerstone in many gene therapies to identify individuals with close genomic data. Among the indicators of genomic similarity, edit distance is regarded as one of the most important and frequently-used metrics for biomedical research [29].

However, genomic information is highly sensitive, thus malicious analysis over genomic data could lead to terrifying privacy leakage. It has been reported that researchers could identify a patient by analyzing his/her genome sequence [32], or infer the close relatives given a target person's gene sequence [19]. To address the privacy issues, several works have been proposed to securely perform SSQ operations on genomic data. Researchers used the secure multi-party computation model (SMC) to perform a query operation over the genomic data set without revealing the query information [2, 3, 35, 36]. However, these works produce a large amount of user-side computations and communications, which are too unwieldy to be practical.

To free users from these burdens, another paradigm that is based on secure computing outsourcing is investigated. In this paradigm, the data owner securely outsources data to the cloud server, which will then process user's queries by searching within the data set in a privacy-preserving manner. Several privacy-preserving protocols [5, 9, 14] are proposed to compute the edit distance between two sequences based on homomorphic encryptions. Though correctness demonstrated, these approaches raise performance concerns primarily due to the length of genomic sequences.

To address this issue, in a recent piece of work [23], Kim et al. developed a secure approximate edit distance scheme aiming at reducing the computation overhead on cloud based on the set-difference metric, which enables parallel processing in computation. However, the query user still suffers from heavy overhead. In addition, all these works heavily rely on homomorphic encryption primitives, which are costly to make these schemes scalable to support multiple data owners and users.

In this work, we study the secure similar sequence query problem over outsourced genomic data. Our goal is to develop a secure SSQ solution that is efficient over large-volume genomic data. To this end, we propose two protocols in the mixed form [12], which combine two-party secret sharing, garbled circuit, and partial homomorphic encryptions together and use them to jointly fulfill the

secure SSQ function. Our protocols could perform k-nearest sequence queries securely by guaranteeing the privacy of outsourced data and queries. To improve our protocols' performance, we employ the lightweight additive secret sharing method to evaluate the majority of secure-computation functions, while reducing the usage of homomorphic encryptions to the minimum. The computational overhead on query users is negligible and the communication overhead is only one-time. In addition, our protocols support multiple query-generating users and data owners, yielding a scalable solution in advancing secure SSQ in practice.

Challenges. We observe that finding the minimum and selecting the branch are key operations in the process of edit distance computation. During these processes, the information regarding the index of the result should not be revealed, because some confidential information, such as the access patterns could be leaked with indices disclosed. Although there are some works tackling these problems, their performance is not satisfactory, since multiple costly encryptions, decryptions, and evaluations on ciphertexts are needed [10, 13]. Also, their schemes didn't consider the scalability issue in multi-user scenarios.

In addition, our mixed protocols utilize different types of sub-protocols, which use different message spaces. For example, in the offline phase of our secure shuffling protocol, a conversion from message space of additive secret sharing to that of Paillier encryption is needed. Though straightforward to convert the message space while guaranteeing the correctness of protocols in evaluating corresponding functions, it is challenging to guarantee security during this process.

Our Contribution. By addressing the aforementioned challenges, we propose two protocols that could securely and efficiently compute k nearest sequences given queries from multiple users. Specifically, our contributions are:

- In combination with multiple secure computation methods including secret sharing, garbled circuits, and homomorphic encryption, we present a set of secure hybrid sub-protocols, which can be used as building blocks for secure similar sequence query.
- Based on our building blocks, we implement two secure SSQ protocols with exact edit distance (SSQ-I) and approximate edit distance (SSQ-II). Our protocols allow secure multi-user SSQ on a joint genomic database collected from different data sources. By shifting a large portion of computational workload including the homomorphic encryptions to the offline phase, we could remarkably improve the query performance.
- We formally prove that our protocols are secure under semi-honest adversaries model. We present an extensive experimental evaluation of the proposed protocols that are implemented on a commercial cloud platform, showing that the proposed methods scale well for large data sets, and clearly outperform state-of-the-art works.

The rest of the paper is organized as follows. Section 2 reviews edit distance computation and some secure computation primitives. Section 3 gives an overview of our system framework. A set of privacy-preserving sub-protocols and their implementations are provided in Section 4. In Section 5, the two proposed SSQ protocols are explained in detail. We analyze the computational complexity and the security of the proposed protocols in Section 6. The proposed protocols are evaluated through extensive simulations in Section 7. We make a discussion and review some related work in Section 8 and Section 9, and make a conclusion in Section 10.

2 PRELIMINARIES

2.1 Edit Distance Protocol for Genomic Sequences

Edit distance is a measure to quantify how dissimilar two strings of characters are to one another by counting the minimum number of edits required to transform one string into the other. Wagner-Fisher algorithm is a common method for computing edit distance based on dynamic programming, we refer the reader to Appendix A for details.

However, the overhead for computing the exact genome edit distance in a privacy-preserving manner is too large because of large datasets and long sequences. There is a unique feature in human genome sequences that two average human individuals are extremely similar at the genetic level (about 99.5%) [36]. So researchers, in recent years, have developed a series of approximate edit distance protocols based on this observation [3, 23, 36]. Next, we briefly describe an advanced approximate algorithm[1] [3].

First, the algorithm converts the edit-distance function (ED) into a block-wise approximation. Each sequence S_i in the database and query Q are broken into t short length (less than 15) blocks $(S_{i,1}, \ldots, S_{i,t})$ and $(Q_1, \ldots, Q_t)$. Then the approximate edit-distance between S_i and Q can be computed as follows:

$$\text{ApproxED}(Q, S_i) \approx \sum_{l=1}^{t} \text{ED}(Q_l, S_{i,l}). \tag{1}$$

For genomic data, since each block has only a few distinct values, the size of the set of values $T_l = \{S_{i,l} : i = 1, \ldots, m\}$ is much smaller than m. Furthermore, there is a high probability that the block Q_l is in the set T_l. Let $v = max(|T_l| : l = 1, \ldots, t)$, $T_l = \{u_{l,1}, \ldots, u_{l,v}\}$, define a bit variable $\chi_{l,j} \leftarrow (u_{l,j} == Q_l ? 1 : 0)$ to indicate whether $u_{l,j}$ is equal to Q_l. In the case where $Q_l \notin T_l$, $\chi_{l,j} = 0$ will bring the error. However, [3] verified that the error rate is minor on real genomic data. Thus, we have

$$\text{ED}(Q_l, S_{i,l}) \approx \sum_{j=1}^{v} \chi_{l,j} \cdot \text{ED}(u_{l,j}, S_{i,l}), \tag{2}$$

then

$$\text{ApproxED}(Q, S_i) \approx \sum_{l=1}^{t} \sum_{j=1}^{v} \chi_{l,j} \cdot \text{ED}(u_{l,j}, S_{i,l}). \tag{3}$$

Note that $\text{ED}(u_{l,j}, S_{i,l})$ is not relevant to the query Q, so it can be pre-computed.

2.2 Secure Computation

2.2.1 Additive Secret Sharing and Multiplication Triplets. For the additive secret sharing [12], an ℓ-bit value x is shared additively in the ring $\mathbb{Z}_{2^\ell}$ as the sum of two values. For an ℓ-bit additive secret sharing $\langle x \rangle$ of x, we have $\langle x \rangle^{\mathcal{A}} + \langle x \rangle^{\mathcal{B}} \equiv x \pmod{2^\ell}$ where $\langle x \rangle^{\mathcal{A}}, \langle x \rangle^{\mathcal{B}} \in \mathbb{Z}_{2^\ell}$ and $\langle x \rangle^{\alpha}$ is only known by party α ($\alpha \in \{\mathcal{A}, \mathcal{B}\}$). We denote a shared value x as $\langle x \rangle$. To recover (Rec$(\cdot, \cdot)$) the value

[1] This work won the 1st place for accuracy and speed in the recent iDASH competition (http://www.humangenomeprivacy.org/2016/).

x, party $\mathcal{A}$ ($\mathcal{B}$) sends $\langle x\rangle^{\mathcal{A}}$ ($\langle x\rangle^{\mathcal{B}}$) to party $\mathcal{B}$ ($\mathcal{A}$) who computes $x = \langle x\rangle^{\mathcal{A}} + \langle x\rangle^{\mathcal{B}}$. The basic operation on additive secret sharing values can be defined as follows:

Addition. To compute the sum of two shared values $\langle x\rangle$ and $\langle y\rangle$, $\langle z\rangle = \langle x+y\rangle = \langle x\rangle + \langle y\rangle$ can be defined as: party α locally computes $\langle z\rangle^{\alpha} = \langle x\rangle^{\alpha} + \langle y\rangle^{\alpha}$. To compute the sum of a shared values $\langle x\rangle$ and a public constant c, $\langle z\rangle = \langle x+c\rangle = \langle x\rangle + c$ can be defined as: party $\mathcal{A}$ locally computes $\langle z\rangle^{\mathcal{A}} = \langle x\rangle^{\mathcal{A}} + c$, party $\mathcal{B}$ locally computes $\langle z\rangle^{\mathcal{B}} = \langle x\rangle^{\mathcal{B}}$.

Multiplication. $\langle z\rangle = \langle x\rangle \cdot \langle y\rangle$: In order to perform multiplication, we take advantage of pre-computed multiplication triplets [4, 12] of the form $\langle c\rangle = \langle a\rangle \cdot \langle b\rangle$: party α locally computes $\langle e\rangle^{\alpha} = \langle x\rangle^{\alpha} - \langle a\rangle^{\alpha}$ and $\langle f\rangle^{\alpha} = \langle y\rangle^{\alpha} - \langle b\rangle^{\alpha}$. Both parties run e=Rec($\langle e\rangle^{\mathcal{A}}$, $\langle e\rangle^{\mathcal{B}}$) and f=Rec($\langle f\rangle^{\mathcal{A}}$,$\langle f\rangle^{\mathcal{B}}$), then party $\mathcal{A}$ sets $\langle z\rangle^{\mathcal{A}} = f\cdot\langle a\rangle^{\mathcal{A}} + e\cdot\langle b\rangle^{\mathcal{A}} + \langle c\rangle^{\mathcal{A}}$ and $\mathcal{B}$ sets $\langle z\rangle^{\mathcal{B}} = e\cdot f + f\cdot\langle a\rangle^{\mathcal{B}} + e\cdot\langle b\rangle^{\mathcal{B}} + \langle c\rangle^{\mathcal{B}}$.

In the remainder of this paper, the additive secret sharing is abbreviated as secret sharing or sharing for brevity.

2.2.2 Yao's Garbled Circuits. Yao's garbled circuits allow two parties holding inputs x and y, respectively, to evaluate an arbitrary function $f(x, y)$ without leaking any information about their inputs beyond what is implied by the function output [25]. Three simple circuits will be used in this paper to construct secure protocols. An ADD circuit takes two integers x and y as inputs and outputs an integer z, such that $z = x + y$. A CMP circuit takes x and y as input, and outputs 1 if $x > y$ and 0 otherwise. An EQ circuit takes x and y as input, and outputs 1 if $x == y$ and 0 otherwise.

2.2.3 Paillier Cryptosystem with Distributed Decryption. Paillier cryptosystem was first proposed by [31], which is an additive homomorphic encryption scheme. Hazay et al. adapted the Paillier cryptosystem to separate private key sk into two shares $sk^{(1)}, sk^{(2)}$ to support distributed decryption [17]. Let $E_{pk}(\cdot)$ and $D_{sk}(\cdot)$ be the encryption and decryption function, where public key pk is given by (N, g) and N is a product of two large primes and g is in $\mathbb{Z}^*_{N^2}$. Also, let $PDec_{sk^{(1)}}(\cdot)$, $PDec_{sk^{(2)}}(\cdot)$ be the partial decryption function with partial private key $sk^{(1)}, sk^{(2)}$. Given $a, b \in \mathbb{Z}_N$, the Paillier cryptosystem with distributed decryption exhibits the following properties:

Distributed Decryption[2]: $a' \leftarrow PDec_{sk^{(1)}}(E_{pk}(a))$ and $a \leftarrow PDec_{sk^{(2)}}(a')$.

Homomorphic Addition: $D_{sk}(E_{pk}(a)\cdot E_{pk}(b) \mod N^2) = (a+b) \mod N$, $D_{sk}(E_{pk}(a)^b \mod N^2) = a\cdot b \mod N$.

3 PROBLEM OVERVIEW

In this section, we formalize the system model, outline the problem statement, and describe the security model. For references, a summary of notations is given in Table 1.

3.1 System Model

As shown in Figure 1, our system model involves two non-colluding servers, multiple data owners and a set of query users.

(1) Two-Servers: We consider the existence of two non-colluding cloud servers, denoted by Server $\mathcal{A}$ and Server $\mathcal{B}$. Both servers have the ability to maintain a database contributed by multiple data owners. The two servers cooperate to answer queries from multiple query users in a privacy-preserving manner.
(2) Data owners: Data are generated or collected by multiple data owners, they would upload the secret sharing data to Server $\mathcal{A}$ and Server $\mathcal{B}$.
(3) Query users: The goal of query users is to request the two-servers to perform some queries over the secret sharing data. After the queries has been performed, the result can be obtained by the requesting users.

[2]The original work presents this process in a different and general form.

Table 1: The Summary of Notations

Notations	Definitions
$\langle x\rangle$	a pair of secret shares of the value x i.e., $(\langle x\rangle^{\mathcal{A}}, \langle x\rangle^{\mathcal{B}})$
$\langle x\rangle^{\mathcal{A}}$ /$\langle x\rangle^{\mathcal{B}}$	the share of x stored in $\mathcal{A}$ or $\mathcal{B}$
pk	public key in Paillier
$sk^{(1)}/sk^{(2)}$	partial private keys in Paillier
$[x]_{pk}$	Encrypted data x under pk
$PDec_{sk^{(i)}}(\cdot)$	Partial decryption with $sk^{(i)}$, $i = 1/2$

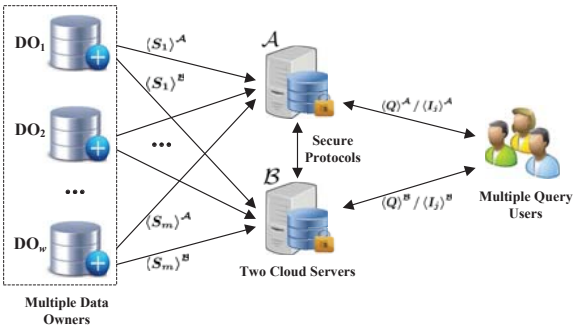


Figure 1: System Model.

3.2 Problem Statement

In this part, we first describe how to encode the genomic sequence in our system and then present the problem statement.

Since genomic sequences are represented by the four letter alphabet of nucleotides {A, C, G, T}, each letter can be represented as an integer and each genomic sequence can be represented as an integer vector. For instance, we can use the integers 0,1,2,3 to represent A,C,G,T, then the sequence "ATCGC" can be expressed as [0,3,1,2,1]. Actually, in our scheme, any arbitrary integer works without influencing the correctness of the result, but thereafter we will use the above encoding scheme for uniformity and the genomic sequences in the remainder of this paper refer to encoded genomic sequences.

We consider w data owners $\mathrm{DO}_1, \cdots, \mathrm{DO}_w$ (e.g. hospitals, research institutions) who agree to construct a joint genomic database

D based on the sequences they possess. For reasons of privacy, each sequence S_i is divided into two shares $\langle S_i \rangle^{\mathcal{A}}$ and $\langle S_i \rangle^{\mathcal{B}}$ locally using additive secret sharing, and then each data owner sends the two shares to server $\mathcal{A}$ and server $\mathcal{B}$, respectively. Consider a query user (e.g. physician) who wants to query the top-k similar genomic sequences corresponding to query Q. Refer to [3, 36], we use edit distance as the metric to measure the similarity between different genomic sequences. In order to preserve the query privacy, the user uses additive secret sharing to partition the query Q into two shares $\langle Q \rangle^{\mathcal{A}}$ and $\langle Q \rangle^{\mathcal{B}}$, then send them to two servers. After receiving the query request, the two servers run a series of secure protocols to compute the edit distances and return the indices of top-k results (i.e., the indices of k-nearest sequences) $I = \{I_j : 1 \leq j \leq k\}$ to the user without revealing any private information about the genomic database D and the query Q.

3.3 Security Model

Adversary Model. We consider the problem of secure similar sequence query under the semi-honest adversary model. That is, we assume that parties are semi-honest (also referred to as honest-but-curious) ones who correctly follow the protocol specification, yet attempt to learn additional information by analyzing the transcript of messages received during the execution. In addition, our work also assumes that the two servers do not collude.

Desired Privacy Properties. Recall that our protocols aim to securely compute edit distance and return the indices of k nearest neighbors to query users. Our protocols should be secure under the semi-honest adversaries model. Our query schemes should protect data privacy, query privacy, and data access patterns. Specially, $\mathcal{A}$ and $\mathcal{B}$ should know nothing about the exact data of D except the database size and sequence length. In addition, $\mathcal{A}$ and $\mathcal{B}$ should know nothing about the query Q except the sequence length. Access patterns to the data, such as the indices of the top-k query results, should not be revealed to $\mathcal{A}$ and $\mathcal{B}$ to prevent any inference attacks.

4 BUILDING BLOCKS

In this section, we present a set of generic sub-protocols that will be used in constructing our proposed protocols in Section 5. Specifically, we realize the following five functions on secret sharing data: 1) secure shuffling; 2) secure branching; 3) secure minimum computing; 4) secure exact edit distance computing; 5) secure sequence comparing. These sub-protocols could also be used in other types of secure computing protocols, e.g., secure k-NN query [13] and trajectory similarity computing [26]. Recall that we assume the existence of two semi-honest servers $\mathcal{A}$ and $\mathcal{B}$ such that the Paillier's partial private key (introduced in Section 2.2.3) $sk^{(1)}$ is known only to server $\mathcal{A}$ and $sk^{(2)}$ only to server $\mathcal{B}$ whereas $pk = (N, g)$ is treated as public. In the following protocols, unless explicitly stated, we assume all operations to be performed are on the ring $\mathbb{Z}_{2^\ell}$. In the following, we first detail the secure shuffling protocol, then realize other functions one-by-one.

4.1 Secure Shuffling (SSF) Protocol

Consider an original sequence $\mathbf{x}$=$[x_1, \ldots, x_n]$ is additively secret-shared into two shares $\langle \mathbf{x} \rangle^{\mathcal{A}} = [\langle x_1 \rangle^{\mathcal{A}}, \ldots, \langle x_n \rangle^{\mathcal{A}}]$ and $\langle \mathbf{x} \rangle^{\mathcal{B}} = [\langle x_1 \rangle^{\mathcal{B}}, \ldots, \langle x_n \rangle^{\mathcal{B}}]$. Secure Shuffling (SSF) protocol is to realize the *function* that permutes the original sequences $\langle \mathbf{x} \rangle^{\mathcal{A}}$ and $\langle \mathbf{x} \rangle^{\mathcal{B}}$ into the new sequences $\langle \mathbf{x}' \rangle^{\mathcal{A}}$ and $\langle \mathbf{x}' \rangle^{\mathcal{B}}$. During this protocol, no information regarding $\mathbf{x}$ is revealed to $\mathcal{A}$ and $\mathcal{B}$. Inspired by the previous work [26], we permute the original sequences $\langle \mathbf{x} \rangle^{\mathcal{A}} = [\langle x_1 \rangle^{\mathcal{A}}, \ldots, \langle x_n \rangle^{\mathcal{A}}]$ and $\langle \mathbf{x} \rangle^{\mathcal{B}} = [\langle x_1 \rangle^{\mathcal{B}}, \ldots, \langle x_n \rangle^{\mathcal{B}}]$ into the new sequences $\langle \mathbf{x}' \rangle^{\mathcal{A}} = [\langle x_{\pi(\pi'(1))} \rangle^{\mathcal{A}}, \ldots, \langle x_{\pi(\pi'(n))} \rangle^{\mathcal{A}}]$ and $\langle \mathbf{x}' \rangle^{\mathcal{B}} = [\langle x_{\pi(\pi'(1))} \rangle^{\mathcal{B}}, \ldots, \langle x_{\pi(\pi'(n))} \rangle^{\mathcal{B}}]$ by two random permutation functions π (only hold by $\mathcal{A}$) and π' (only hold by $\mathcal{B}$). As long as $\mathcal{A}$ and $\mathcal{B}$ do not know each other's permutation function, neither of them could recover the original indices of $\langle \mathbf{x} \rangle$. In consideration of performance, SSF protocol is divided into two phases: offline and online. The offline phase generates some common assistant values which are independent of the input of protocol, while online phase can achieve the shuffle in a single interaction between $\mathcal{A}$ and $\mathcal{B}$ with the help of the assistant values.

Algorithm 1 shows the main steps in the offline phase. Note that in this phase, the operations are performed on the group $\mathbb{Z}_N$. At the beginning of this protocol, Server $\mathcal{A}$ chooses n random integers $u_1, \ldots, u_n \in \mathbb{Z}_{2^\ell}$ to form a vector $\mathbf{u}$ and chooses n random integers $r_1, \ldots, r_n \in \{0 \cdot 2^\ell, \cdots, (K-2) \cdot 2^\ell\} (K = \lfloor N/2^\ell \rfloor)$ to form a vector $\mathbf{r}$. In this step, we make a conversion from message space of additive secret sharing to that of Paillier encryption by introducing a group of random values of r_i. Then, $\mathcal{A}$ constructs a sequence $L_0 \leftarrow [E_{pk}(u_1 + r_1), \ldots, E_{pk}(u_n + r_n)]$ where $u_i + r_i$ is encrypted under the public key pk, and send it to $\mathcal{B}$. Thanks to the property of distributed decryption in the improved Paillier cryptosystem [17], $\mathcal{B}$ is unable to decrypt L_0 without the help of $\mathcal{A}$. Upon receiving L_0, $\mathcal{B}$ generates a random vector $\mathbf{v}$= $[v_1, \ldots, v_n] (v_i \in \mathbb{Z}_{2^\ell})$ and randomizes L_0 by $\mathbf{v}$ for obtaining $L_1 \leftarrow [E_{pk}(u_1 + v_1 + r_1), \ldots, E_{pk}(u_n + v_n + r_n)]$. After that, $\mathcal{B}$ permutes L_1 using a random permutation function π' and send it to $\mathcal{A}$. Similarly, a random permutation function π is selected by $\mathcal{A}$ to permute L_1. Then, $\mathcal{A}$ uses partial private key $sk^{(1)}$ to partially decrypt L_1 and sends the partial ciphertext L_2 to $\mathcal{B}$. Finally, $\mathcal{B}$ decrypts L_2 by $sk^{(2)}$ to get $L_3 = [u_{\pi(\pi'(1))} + v_{\pi(\pi'(1))} + r_{\pi(\pi'(1))}, \ldots, u_{\pi(\pi'(n))} + v_{\pi(\pi'(n))} + r_{\pi(\pi'(n))}]$, and then eliminates $r_{\pi(\pi'(i))}$ by modulus operations for getting $\langle \mathbf{x}' \rangle^{\mathcal{B}} = -(L_3 \bmod 2^\ell) = [-u_{\pi(\pi'(1))} - v_{\pi(\pi'(1))}, \ldots, -u_{\pi(\pi'(n))} - v_{\pi(\pi'(n))}]$.

Next, we describe how to implement the online phase of SSF protocol in the Algorithm 2. The input in this phase is a secret sharing sequence $\langle \mathbf{x} \rangle$ i.e., $\mathcal{A}$ inputs $\langle \mathbf{x} \rangle^{\mathcal{A}} = [\langle x_1 \rangle^{\mathcal{A}}, \cdots, \langle x_n \rangle^{\mathcal{A}}]$ and $\mathcal{B}$ inputs $\langle \mathbf{x} \rangle^{\mathcal{B}} = [\langle x_1 \rangle^{\mathcal{B}}, \cdots, \langle x_n \rangle^{\mathcal{B}}]$. Note that at this point, $\mathcal{A}$ holds $\pi, \mathbf{u}$ and $\mathcal{B}$ holds $\pi', \mathbf{v}$, where $\mathbf{u}, \mathbf{v}$ are the assistant values. To start with, $\mathcal{A}$ additively masks each $\langle x_i \rangle^{\mathcal{A}}$ with u_i and assembles them in the sequence L_4, such that $L_4 \leftarrow [\langle x_1 \rangle^{\mathcal{A}} + u_1, \ldots, \langle x_n \rangle^{\mathcal{A}} + u_n]$. This prevents $\mathcal{B}$ from learning $\langle x_i \rangle^{\mathcal{A}}$. After receiving L_4, $\mathcal{B}$ accumulates $L_4, \langle \mathbf{x} \rangle^{\mathcal{B}}, \mathbf{v}$ in an element-wise manner to get $L_5 = [x_1 + u_1 + v_1, \ldots, x_n + u_n + v_n]$. Then, L_5 is permuted by π' and sent back to $\mathcal{A}$. Finally, $\mathcal{A}$ uses π to permutate L_5 again. So the protocol ends with $\mathcal{A}$ holding $\langle \mathbf{x}' \rangle^{\mathcal{A}} = [x_{\pi(\pi'(1))} + u_{\pi(\pi'(1))} + v_{\pi(\pi'(1))}, \ldots, x_{\pi(\pi'(n))} + u_{\pi(\pi'(n))} + v_{\pi(\pi'(n))}]$ and $\mathcal{B}$ holding $\langle \mathbf{x}' \rangle^{\mathcal{B}} = [-u_{\pi(\pi'(1))} - v_{\pi(\pi'(1))}, \ldots, -u_{\pi(\pi'(n))} - v_{\pi(\pi'(n))}]$, i.e., the permuted sequence $\langle \mathbf{x}' \rangle$.

We should note that the performance of our protocol in the offline phase can be improved by a universal data packing technology, we refer the reader to [6] for details. As for the online part, our protocol

Algorithm 1 The Offline Phase of Secure Shuffling Protocol

Input: None ($\mathcal{A}$ holds $pk, sk^{(1)}$; $\mathcal{B}$ holds $pk, sk^{(2)}$)
Output: $\mathcal{B}$ outputs $\langle \mathbf{x}' \rangle^{\mathcal{B}} = [\langle x_1' \rangle^{\mathcal{B}}, \cdots, \langle x_n' \rangle^{\mathcal{B}}]$

$\mathcal{A}$:
  Pick n random integers $u_1, \ldots, u_n \in \mathbb{Z}_{2^\ell}$, $\mathbf{u} = [u_1, \ldots, u_n]$
  Pick n random integers $r_1, \ldots, r_n \in \{0 \cdot 2^\ell, 1 \cdot 2^\ell, \cdots, (K-2) \cdot 2^\ell\} (K = \lfloor N/2^\ell \rfloor)$, $\mathbf{r} = [r_1, \ldots, r_n]$
  $L_0 \leftarrow [E_{pk}(u_1 + r_1), \ldots, E_{pk}(u_n + r_n)]$
  Send L_0 to $\mathcal{B}$
$\mathcal{B}$:
  Receive L_0 from $\mathcal{A}$
  Pick n random integers $v_1, \ldots, v_n \in \mathbb{Z}_{2^\ell}$
  $\mathbf{v} = [v_1, \ldots, v_n]$
  $L_1 \leftarrow [E_{pk}(u_1 + r_1) \cdot E_{pk}(v_1), \ldots, E_{pk}(u_n + r_n) \cdot E_{pk}(v_n)]$
  $= [E_{pk}(u_1 + v_1 + r_1), \ldots, E_{pk}(u_n + v_n + r_n)]$
  Generate a random π', then permute L_1 by π':
  $L_1 \leftarrow [E_{pk}(u_{\pi'(1)} + v_{\pi'(1)} + r_{\pi'(1)}), \ldots, E_{pk}(u_{\pi'(n)} + v_{\pi'(n)} + r_{\pi'(n)})]$
  Send L_1 to $\mathcal{A}$
$\mathcal{A}$:
  Receive L_1 from $\mathcal{B}$
  Generate a random π, then permute L_1 by π:
  $L_1 \leftarrow [E_{pk}(u_{\pi(\pi'(1))} + v_{\pi(\pi'(1))} + r_{\pi(\pi'(1))}), \ldots, E_{pk}(u_{\pi(\pi'(n))} + v_{\pi(\pi'(n))} + r_{\pi(\pi'(n))})]$
  Partial decrypt L_1 with $PDec_{sk^{(1)}}(\cdot)$ to obtain L_2
  Send L_2 to $\mathcal{B}$
$\mathcal{B}$:
  Receive L_2 from $\mathcal{A}$
  Decrypt L_2 with $PDec_{sk^{(2)}}(\cdot)$ to obtain:
  $L_3 \leftarrow [u_{\pi(\pi'(1))} + v_{\pi(\pi'(1))} + r_{\pi(\pi'(1))}, \cdots, u_{\pi(\pi'(n))} + v_{\pi(\pi'(n))} + r_{\pi(\pi'(n))}]$
  $\langle \mathbf{x}' \rangle^{\mathcal{B}} \leftarrow [\langle x_1' \rangle^{\mathcal{B}}, \cdots, \langle x_n' \rangle^{\mathcal{B}}] = -(L_3 \mod 2^\ell)$
  $= [-u_{\pi(\pi'(1))} - v_{\pi(\pi'(1))}, \ldots, -u_{\pi(\pi'(n))} - v_{\pi(\pi'(n))}]$

enjoys excellent performance since it only requires one interaction and involves no cryptographic operation.

4.2 Secure Branching (SBC) Protocol

We assume that $\mathcal{A}$ inputs a set of shared values $\langle x_1 \rangle^{\mathcal{A}}$, $\langle x_2 \rangle^{\mathcal{A}}$, $\langle y_1 \rangle^{\mathcal{A}}$, $\langle y_2 \rangle^{\mathcal{A}}$ and $\mathcal{B}$ inputs another part $\langle x_1 \rangle^{\mathcal{B}}$, $\langle x_2 \rangle^{\mathcal{B}}$, $\langle y_1 \rangle^{\mathcal{B}}$, $\langle y_2 \rangle^{\mathcal{B}}$. Secure Branching (SBC) protocol is to realize the *function* that securely executes the following conditional statement on secret sharing data:

$$\textbf{if}(x_1 > x_2) \textbf{ then } y \leftarrow y_1 \textbf{ else } y \leftarrow y_2.$$

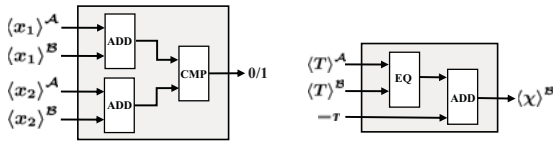
The output of the protocol is in the form of secret sharing, i.e., $\mathcal{A}$ outputs $\langle y \rangle^{\mathcal{A}}$ and $\mathcal{B}$ outputs $\langle y \rangle^{\mathcal{B}}$. During this process, no information regarding x_1, x_2, y_1, and y_2 is revealed to Server $\mathcal{A}$ and Server $\mathcal{B}$. Here, we construct an efficient mixed SBC protocol based on the SSF protocol. The main steps involved in SBC protocol are shown in Algorithm 3. By the SSF protocol, Server $\mathcal{A}$ and Server $\mathcal{B}$ jointly use the permutation function $\pi(\pi'(\cdot))$ to shuffle the secret sharing sequences $[\langle x_1 \rangle, \langle x_2 \rangle]$ to get a new one $[\langle x_{\pi(\pi'(1))} \rangle, \langle x_{\pi(\pi'(2))} \rangle]$. Then, the same permutation function $\pi(\pi'(\cdot))$ is applied to $[\langle y_1 \rangle, \langle y_2 \rangle]$ for obtaining $[\langle y_{\pi(\pi'(1))} \rangle, \langle y_{\pi(\pi'(2))} \rangle]$. After shuffling, a ADD-CMP circuit is used to compare $\langle x_{\pi(\pi'(1))} \rangle$ and $\langle x_{\pi(\pi'(2))} \rangle$. As depicted in Figure 2(a), the anatomy of ADD-CMP is very simple as it only consists of two ADD

Algorithm 2 The Online Phase of Secure Shuffling Protocol

Input: $\mathcal{A}$ inputs $\langle \mathbf{x} \rangle^{\mathcal{A}} = [\langle x_1 \rangle^{\mathcal{A}}, \cdots, \langle x_n \rangle^{\mathcal{A}}]$($\mathcal{A}$ holds $\pi, \mathbf{u}$)
  $\mathcal{B}$ inputs $\langle \mathbf{x} \rangle^{\mathcal{B}} = [\langle x_1 \rangle^{\mathcal{B}}, \cdots, \langle x_n \rangle^{\mathcal{B}}]$ ($\mathcal{B}$ holds $\pi', \mathbf{v}$)
Output: $\mathcal{A}$ outputs $\langle \mathbf{x}' \rangle^{\mathcal{A}} = [\langle x_1' \rangle^{\mathcal{A}}, \cdots, \langle x_n' \rangle^{\mathcal{A}}]$

$\mathcal{A}$:
  $L_4 \leftarrow [\langle x_1 \rangle^{\mathcal{A}}, \ldots, \langle x_n \rangle^{\mathcal{A}}]$
  Mask L_4 by $\mathbf{u}$ to get:
  $L_4 \leftarrow [\langle x_1 \rangle^{\mathcal{A}} + u_1, \ldots, \langle x_n \rangle^{\mathcal{A}} + u_n]$
  Send L_4 to $\mathcal{B}$
$\mathcal{B}$:
  Receive L_4 from $\mathcal{A}$
  Compute $L_5 \leftarrow L_4 + \langle \mathbf{x} \rangle^{\mathcal{B}} + \mathbf{v}$ to get
  $L_5 = [x_1 + u_1 + v_1, \ldots, x_n + u_n + v_n]$
  Permute L_5 by π':
  $L_5 \leftarrow [x_{\pi'(1)} + u_{\pi'(1)} + v_{\pi'(1)}, \ldots, x_{\pi'(n)} + u_{\pi'(n)} + v_{\pi'(n)}]$
  Send L_5 to $\mathcal{A}$
$\mathcal{A}$:
  Receive L_5 from $\mathcal{B}$
  Permute L_5 by π to obtain
  $\langle \mathbf{x}' \rangle^{\mathcal{A}} = [\langle x_1' \rangle^{\mathcal{A}}, \cdots, \langle x_n' \rangle^{\mathcal{A}}] = [x_{\pi(\pi'(1))} + u_{\pi(\pi'(1))} + v_{\pi(\pi'(1))}, \ldots, x_{\pi(\pi'(n))} + u_{\pi(\pi'(n))} + v_{\pi(\pi'(n))}]$

circuits and a CMP circuit. One ADD circuit takes $\langle x_{\pi(\pi'(1))} \rangle^{\mathcal{A}}$ and $\langle x_{\pi(\pi'(1))} \rangle^{\mathcal{B}}$ as inputs while the other takes $\langle x_{\pi(\pi'(2))} \rangle^{\mathcal{A}}$ and $\langle x_{\pi(\pi'(2))} \rangle^{\mathcal{B}}$ as inputs, then the two outputs serve as the inputs of the CMP circuit. In this way, ADD-CMP outputs 1 if $x_{\pi(\pi'(1))} > x_{\pi(\pi'(2))}$ and 0 otherwise. Specifically, the circuit evaluator runs oblivious transfer (OT) protocol with the circuit constructor to obliviously obtain the garbled input corresponding to its private input, then evaluates the garbled circuit to get the final result. If the result is public, the evaluator directly sends it to the constructor. Note that in this protocol, $\mathcal{A}$ or $\mathcal{B}$ can be either the circuit constructor or the circuit evaluator and the output θ is public to $\mathcal{A}$ and $\mathcal{B}$. Finally, based on the output of the ADD-CMP circuit, Server $\mathcal{A}$ and Server $\mathcal{B}$ separately determine the final values $\langle y \rangle^{\mathcal{A}}$ and $\langle y \rangle^{\mathcal{B}}$, i.e., if $\theta == 1$ then $\langle y \rangle^{\alpha} \leftarrow \langle y_{\pi(\pi'(1))} \rangle^{\alpha}$ else $\langle y \rangle^{\alpha} \leftarrow \langle y_{\pi(\pi'(2))} \rangle^{\alpha} (\alpha \in \{\mathcal{A}, \mathcal{B}\})$.



(a) ADD-CMP (b) EQ-ADD

Figure 2: The structure of circuits

4.3 Secure Minimum Selection (SMS) Protocol

Let us assume that $\mathcal{A}$ inputs $\langle \mathbf{x} \rangle^{\mathcal{A}} = [\langle x_1 \rangle^{\mathcal{A}}, \ldots, \langle x_n \rangle^{\mathcal{A}}]$ and $\mathcal{B}$ inputs $\langle \mathbf{x} \rangle^{\mathcal{B}} = [\langle x_1 \rangle^{\mathcal{B}}, \ldots, \langle x_n \rangle^{\mathcal{B}}]$. Secure Minimum Selection (SMS) protocol is to realize the *function* that $\mathcal{A}$ outputs $\langle x_{min} \rangle^{\mathcal{A}}$ and $\mathcal{B}$ outputs $\langle x_{min} \rangle^{\mathcal{B}}$, where $x_{min} = \min(x_1, \cdots, x_n)$. In the process, no information regarding the values of $\mathbf{x} = [x_1, \cdots, x_n]$ is revealed to $\mathcal{A}$ and $\mathcal{B}$. Here, we construct an efficient mixed SMS protocol based on the SSF protocol. The main steps involved in the

Algorithm 3 Secure Branching Protocol

Input: $\mathcal{A}$ inputs $[\langle x_1\rangle^{\mathcal{A}}, \langle x_2\rangle^{\mathcal{A}}], [\langle y_1\rangle^{\mathcal{A}}, \langle y_2\rangle^{\mathcal{A}}]$;
$\mathcal{B}$ inputs $[\langle x_1\rangle^{\mathcal{B}}, \langle x_2\rangle^{\mathcal{B}}], [\langle y_1\rangle^{\mathcal{B}}, \langle y_2\rangle^{\mathcal{B}}]$;
Output: $\mathcal{A}$ outputs $\langle y\rangle^{\mathcal{A}}$; $\mathcal{B}$ outputs $\langle y\rangle^{\mathcal{B}}$

1: $\underline{\mathcal{A} \text{ and } \mathcal{B}}$:
2: $[\langle x_{\pi(\pi'(1))}\rangle, \langle x_{\pi(\pi'(2))}\rangle] \leftarrow \text{SSF}([\langle x_1\rangle, \langle x_2\rangle])$
3: $[\langle y_{\pi(\pi'(1))}\rangle, \langle y_{\pi(\pi'(2))}\rangle] \leftarrow \text{SSF}([\langle y_1\rangle, \langle y_2\rangle])$
4: $\theta \leftarrow \text{ADD-CMP}(\langle x_{\pi(\pi'(1))}\rangle, \langle x_{\pi(\pi'(2))}\rangle)$
5: **if** $\theta == 1$ **then**
6: $\underline{\mathcal{A}}$: $\langle y\rangle^{\mathcal{A}} \leftarrow \langle y_{\pi(\pi'(1))}\rangle^{\mathcal{A}}$; $\underline{\mathcal{B}}$: $\langle y\rangle^{\mathcal{B}} \leftarrow \langle y_{\pi(\pi'(1))}\rangle^{\mathcal{B}}$
7: **else**
8: $\underline{\mathcal{A}}$: $\langle y\rangle^{\mathcal{A}} \leftarrow \langle y_{\pi(\pi'(2))}\rangle^{\mathcal{A}}$; $\underline{\mathcal{B}}$: $\langle y\rangle^{\mathcal{B}} \leftarrow \langle y_{\pi(\pi'(2))}\rangle^{\mathcal{B}}$

Algorithm 4 Secure Minimum Selection Protocol

Input: $\mathcal{A}$ inputs $\langle \mathbf{x}\rangle^{\mathcal{A}} = [\langle x_1\rangle^{\mathcal{A}}, \cdots, \langle x_n\rangle^{\mathcal{A}}]$
$\mathcal{B}$ inputs $\langle \mathbf{x}\rangle^{\mathcal{B}} = [\langle x_1\rangle^{\mathcal{B}}, \cdots, \langle x_n\rangle^{\mathcal{B}}]$
Output: $\mathcal{A}$ outputs $\langle x_{min}\rangle^{\mathcal{A}}$; $\mathcal{B}$ outputs $\langle x_{min}\rangle^{\mathcal{B}}$

1: $\underline{\mathcal{A} \text{ and } \mathcal{B}}$:
2: $\langle \mathbf{x}'\rangle \leftarrow \text{SSF}(\langle \mathbf{x}\rangle) = [\langle x_{\pi(\pi'(1))}\rangle, \cdots, \langle x_{\pi(\pi'(n))}\rangle]$
3: $\underline{\mathcal{A}}$: $\langle x_\delta\rangle^{\mathcal{A}} \leftarrow \langle x_{\pi(\pi'(1))}\rangle^{\mathcal{A}}$
4: $\underline{\mathcal{B}}$: $\langle x_\delta\rangle^{\mathcal{B}} \leftarrow \langle x_{\pi(\pi'(1))}\rangle^{\mathcal{B}}$
5: **for** $1 \le i \le n-1$ **do**
6: $\underline{\mathcal{A} \text{ and } \mathcal{B}}$: $\theta \leftarrow \text{ADD-CMP}(\langle x_\delta\rangle, \langle x_{\pi(\pi'(i+1))}\rangle)$
7: **if** $\theta == 1$ **then**
8: $\underline{\mathcal{A}}$: $\langle x_\delta\rangle^{\mathcal{A}} \leftarrow \langle x_{\pi(\pi'(i+1))}\rangle^{\mathcal{A}}$
9: $\underline{\mathcal{B}}$: $\langle x_\delta\rangle^{\mathcal{B}} \leftarrow \langle x_{\pi(\pi'(i+1))}\rangle^{\mathcal{B}}$
10: $\underline{\mathcal{A}}$: $\langle x_{min}\rangle^{\mathcal{A}} \leftarrow \langle x_\delta\rangle^{\mathcal{A}}$
11: $\underline{\mathcal{B}}$: $\langle x_{min}\rangle^{\mathcal{B}} \leftarrow \langle x_\delta\rangle^{\mathcal{B}}$

SMS protocol are shown in Algorithm 4. Initially, we utilize SSF protocol to permutate the original sequence to get the new one $\langle \mathbf{x}'\rangle$. After shuffling, $\mathcal{A}$ or $\mathcal{B}$ prepares a garbled circuit ADD-CMP for comparing two additively secret sharing values, e.g. $\langle x_1\rangle$ and $\langle x_2\rangle$. Clearly, the ADD-CMP circuit outputs 1 if $x_1 > x_2$ and 0 otherwise. Same as above, the output of this circuit is public in this protocol. By calling ADD-CMP $n-1$ times, we are able to get the minimum $\langle x_\delta\rangle$. At the end, the output of SMS is $\langle x_{min}\rangle^{\mathcal{A}} \leftarrow \langle x_\delta\rangle^{\mathcal{A}}$ and $\langle x_{min}\rangle^{\mathcal{B}} \leftarrow \langle x_\delta\rangle^{\mathcal{B}}$. In this protocol, the index δ does not reveal any information regarding the position of the minimum in the original sequence to $\mathcal{A}$ and $\mathcal{B}$ as the sequence has been shuffled securely.

4.4 Secure Exact Edit Distance Computation (SEED) Protocol

We assume that $\mathcal{A}$ and $\mathcal{B}$ input two secret sharing sequences $\langle \mathbf{x}\rangle = [\langle x_1\rangle, \cdots, \langle x_{n_1}\rangle]$ and $\langle \mathbf{y}\rangle = [\langle y_1\rangle, \cdots, \langle y_{n_2}\rangle]$, i.e., $\mathcal{A}$ inputs $\langle \mathbf{x}\rangle^{\mathcal{A}}$ and $\langle \mathbf{y}\rangle^{\mathcal{A}}$ while $\mathcal{B}$ inputs $\langle \mathbf{x}\rangle^{\mathcal{B}}$ and $\langle \mathbf{y}\rangle^{\mathcal{B}}$. Secure Exact Edit Distance (SEED) protocol is to realize the *function* that computes exact edit distance $\langle d_{ED}\rangle$ (in the secret sharing form) between $\mathbf{x}$ and $\mathbf{y}$ without revealing any private information about them. Now, we explain how to run Wagner-Fisher algorithm (see Algorithm 9) for edit distance computation in a privacy-preserving manner.

Algorithm 5 Secure Exact Edit Distance Protocol

Input: $\mathcal{A}$ inputs $\langle \mathbf{x}\rangle^{\mathcal{A}} = [\langle x_1\rangle^{\mathcal{A}}, \cdots, \langle x_{n_1}\rangle^{\mathcal{A}}]$ and
$\langle \mathbf{y}\rangle^{\mathcal{A}} = [\langle y_1\rangle^{\mathcal{A}}, \cdots, \langle y_{n_2}\rangle^{\mathcal{A}}]$
$\mathcal{B}$ inputs $\langle \mathbf{x}\rangle^{\mathcal{B}} = [\langle x_1\rangle^{\mathcal{B}}, \cdots, \langle x_{n_1}\rangle^{\mathcal{B}}]$ and
$\langle \mathbf{y}\rangle^{\mathcal{B}} = [\langle y_1\rangle^{\mathcal{B}}, \cdots, \langle y_{n_2}\rangle^{\mathcal{B}}]$
Output: $\mathcal{A}$ outputs $\langle d_{ED}\rangle^{\mathcal{A}}$; $\mathcal{B}$ outputs $\langle d_{ED}\rangle^{\mathcal{B}}$

1: **for** $0 \le i \le n_1$ **do**
2: $\underline{\mathcal{A}}$: $\langle d_{i,0}\rangle^{\mathcal{A}} \leftarrow i$; $\underline{\mathcal{B}}$: $\langle d_{i,0}\rangle^{\mathcal{B}} \leftarrow 0$
3: **for** $0 \le j \le n_2$ **do**
4: $\underline{\mathcal{A}}$: $\langle d_{0,j}\rangle^{\mathcal{A}} \leftarrow j$; $\underline{\mathcal{B}}$: $\langle d_{0,j}\rangle^{\mathcal{B}} \leftarrow 0$
5: $\underline{\mathcal{A}}$:
6: $\langle c_{del}\rangle^{\mathcal{A}} \leftarrow 1, \langle c_{ins}\rangle^{\mathcal{A}} \leftarrow 1, \langle z_0\rangle^{\mathcal{A}} \leftarrow 0, \langle z_1\rangle^{\mathcal{A}} \leftarrow 1$
7: $\underline{\mathcal{B}}$:
8: $\langle c_{del}\rangle^{\mathcal{B}} \leftarrow 0, \langle c_{ins}\rangle^{\mathcal{B}} \leftarrow 0, \langle z_0\rangle^{\mathcal{B}} \leftarrow 0, \langle z_1\rangle^{\mathcal{B}} \leftarrow 0$
9: **for** $1 \le i \le n_1, 1 \le j \le n_2$ **do**
10: $\underline{\mathcal{A}}$: $\langle t_1\rangle^{\mathcal{A}} \leftarrow \langle y_i\rangle^{\mathcal{A}} + 1, \langle t_2\rangle^{\mathcal{A}} \leftarrow \langle y_i\rangle^{\mathcal{A}} - 1$
11: $\underline{\mathcal{B}}$: $\langle t_1\rangle^{\mathcal{B}} \leftarrow \langle y_i\rangle^{\mathcal{B}}, \langle t_2\rangle^{\mathcal{B}} \leftarrow \langle y_i\rangle^{\mathcal{B}}$
12: $\underline{\mathcal{A} \text{ and } \mathcal{B}}$:
13: $\langle t_3\rangle \leftarrow \text{SBC}(\langle t_1\rangle, \langle x_i\rangle, \langle z_0\rangle, \langle z_1\rangle)$
14: $\langle c_{sub}\rangle \leftarrow \text{SBC}(\langle x_i\rangle, \langle t_2\rangle, \langle t_3\rangle, \langle z_1\rangle)$
15: $\langle d_{i,j}\rangle \leftarrow \text{SMS}(\langle d_{i-1,j}\rangle + \langle c_{del}\rangle, \langle d_{i,j-1}\rangle + \langle c_{ins}\rangle, \langle d_{i-1,j-1}\rangle + \langle c_{sub}\rangle)$
16: $\underline{\mathcal{A}}$: $\langle d_{ED}\rangle^{\mathcal{A}} \leftarrow \langle d_{n_1,n_2}\rangle^{\mathcal{A}}$
17: $\underline{\mathcal{B}}$: $\langle d_{ED}\rangle^{\mathcal{B}} \leftarrow \langle d_{n_1,n_2}\rangle^{\mathcal{B}}$

The main steps are shown in Algorithm 5. To start with, server $\mathcal{A}$ and server $\mathcal{B}$ initialize $\langle d_{i,0}\rangle\,(0 \le i \le n_1)$, $\langle d_{0,j}\rangle\,(0 \le j \le n_2)$, $\langle c_{ins}\rangle$ and $\langle c_{del}\rangle$ corresponding to line $1-8$ in Algorithm 5. Then, we begin to calculate $d_{i,j}(1 \le i \le n_1, 1 \le j \le n_2)$ through iterative computing. Note that 2^ℓ should be larger than $\max(n_1, n_2)$ to guarantee the correctness of edit distance computation while ℓ is the bit length of data. In each iteration, $\langle c_{sub}\rangle$ is assigned to 0 or 1 according to whether x_i is equal to y_j, i.e., $c_{sub} \leftarrow (x_i == y_j)?0:1$. In order to have an assignment of $\langle c_{sub}\rangle$ without revealing information regarding x_i and y_j, we first convert "==" operator to ">"operator as follows:

$$c_{sub} \leftarrow (x_i == y_j)?0:1 \Leftrightarrow \begin{cases} t_1 \leftarrow y_j + 1 \\ t_2 \leftarrow y_j - 1 \\ t_3 \leftarrow (t_1 > x_i)?0:1 \\ c_{sub} \leftarrow (x_i > t_2)?t_3:1 \end{cases}$$

then utilize SBC protocol to perform a secure assignment that corresponds to line $12-15$ in Algorithm 5. Note that, the addition in line 15 of Algorithm 5 refers to the addition in the form of additive secret sharing as described in Section 2.2. At the end of each iteration, $\langle d_{i,j}\rangle$ can be calculated by SMS protocol. With $n_1 n_2$ iterations, SEED protocol would return the final edit distance $\langle d_{n_1,n_2}\rangle$.

4.5 Secure Approximate Genomic Sequence Comparison (SAGSC) Protocol

Assume that $\mathcal{A}$ and $\mathcal{B}$ input two secret sharing genomic sequences $\langle \mathbf{x}\rangle = [\langle x_1\rangle, \cdots, \langle x_n\rangle]$ and $\langle \mathbf{y}\rangle = [\langle y_1\rangle, \cdots, \langle y_n\rangle]$, i.e., $\mathcal{A}$ inputs

Algorithm 6 Secure Approximate Genomic Sequence Comparison Protocol

Input: $\mathcal{A}$ inputs $\langle \mathbf{x} \rangle^{\mathcal{A}} = [\langle x_1 \rangle^{\mathcal{A}}, \cdots, \langle x_n \rangle^{\mathcal{A}}]$ and $\langle \mathbf{y} \rangle^{\mathcal{A}} = [\langle y_1 \rangle^{\mathcal{A}}, \cdots, \langle y_n \rangle^{\mathcal{A}}]$
$\mathcal{B}$ inputs $\langle \mathbf{x} \rangle^{\mathcal{B}} = [\langle x_1 \rangle^{\mathcal{B}}, \cdots, \langle x_n \rangle^{\mathcal{B}}]$ and $\langle \mathbf{y} \rangle^{\mathcal{B}} = [\langle y_1 \rangle^{\mathcal{B}}, \cdots, \langle y_n \rangle^{\mathcal{B}}]$
Output: $\mathcal{A}$ outputs $\langle \chi \rangle^{\mathcal{A}}$; $\mathcal{B}$ outputs $\langle \chi \rangle^{\mathcal{B}}$

1: $\mathcal{A}$:
2: $\langle Sum_x \rangle^{\mathcal{A}} \leftarrow \langle x_1 \rangle^{\mathcal{A}} + \cdots + \langle x_n \rangle^{\mathcal{A}}$
3: $\langle Sum_y \rangle^{\mathcal{A}} \leftarrow \langle y_1 \rangle^{\mathcal{A}} + \cdots + \langle y_n \rangle^{\mathcal{A}}$
4: $\langle T \rangle^{\mathcal{A}} \leftarrow \langle Sum_x \rangle^{\mathcal{A}} - \langle Sum_y \rangle^{\mathcal{A}}$
5: Select a random integer $r \in \mathbb{Z}_{2^\ell}$, set $\langle \chi \rangle^{\mathcal{A}} \leftarrow r$
6: $\mathcal{B}$:
7: $\langle Sum_x \rangle^{\mathcal{B}} \leftarrow \langle x_1 \rangle^{\mathcal{B}} + \cdots + \langle x_n \rangle^{\mathcal{B}}$
8: $\langle Sum_y \rangle^{\mathcal{B}} \leftarrow \langle y_1 \rangle^{\mathcal{B}} + \cdots + \langle y_n \rangle^{\mathcal{B}}$
9: $\langle T \rangle^{\mathcal{B}} \leftarrow \langle Sum_y \rangle^{\mathcal{B}} - \langle Sum_x \rangle^{\mathcal{B}}$
10: $\mathcal{A}$ and $\mathcal{B}$:
11: $\langle \chi \rangle^{\mathcal{B}} \leftarrow$ EQ-ADD$(\langle T \rangle^{\mathcal{A}}, \langle T \rangle^{\mathcal{B}}, -r)$

$\langle \mathbf{x} \rangle^{\mathcal{A}}$ and $\langle \mathbf{y} \rangle^{\mathcal{A}}$ while $\mathcal{B}$ inputs $\langle \mathbf{x} \rangle^{\mathcal{B}}$ and $\langle \mathbf{y} \rangle^{\mathcal{B}}$. Secure Approximate Genomic Sequence Comparison (SAGSC) protocol is to realize the *function* that outputs an indicator $\langle \chi \rangle$ where $\chi = 1$ if $\mathbf{x}$ is almost the same as $\mathbf{y}$ and $\chi = 0$ otherwise. During this process, no information regarding $\mathbf{x}$ and $\mathbf{y}$ is revealed to $\mathcal{A}$ and $\mathcal{B}$. A naive method of sequence comparison is to do bit-by-bit comparisons, but generates too much time overhead since a large number of comparison operations are needed for the long genomic sequences. This led us to explore an approximate solution for better efficiency. In this work, we use different specified integers (i.e., 0, 1, 2, 3) to encode four nucleotides (i.e., A, C, G, T), and we observed that individual differences in the human genome sequences are very small. So we conclude that if the sums of the encodings of genomic sequences are equal, they can be regarded as approximately identical sequences. That is, if the equation $\sum_{i=1}^{n} x_i == \sum_{i=1}^{n} y_i$ holds, then two sequences $\langle \mathbf{x} \rangle$ and $\langle \mathbf{y} \rangle$ are treated as the same. In the secret sharing form, the above equation can be converted to

$$\sum_{i=1}^{n} \langle x_i \rangle^{\mathcal{A}} - \sum_{i=1}^{n} \langle y_i \rangle^{\mathcal{B}} == \sum_{i=1}^{n} \langle y_i \rangle^{\mathcal{A}} - \sum_{i=1}^{n} \langle x_i \rangle^{\mathcal{B}}.$$

The overall steps in SAGSC protocol are shown in Algorithm 6. We achieve the comparison and secret sharing of the result by the circuit EQ-ADD (as shown in Figure 2(b)) in which $\mathcal{A}$ is the circuit constructor and the output $\langle \chi \rangle^{\mathcal{B}}$ is only given to $\mathcal{B}$. We will show the error rate of our approximate protocol over a real-world dataset is very small in Section 7.2, and discuss how to further reduce the error rate in Section 8.1.

5 SECURE SIMILAR SEQUENCE QUERY ON GENOMIC DATA

With the above building blocks, we are ready to present the detailed scheme for secure similar sequence query on genomic data. We first propose a secure scheme based on exact edit distance (denoted by SSQ-I), and then present our second scheme, a more efficient solution with the approximate edit distance (denoted by SSQ-II).

5.1 SSQ-I (with the Exact Edit Distance)

Our scheme consists of the following four stages: genomic data outsourcing, query request issuing, secure query executing and result recovering.

5.1.1 Stage 1: Genomic Data Outsourcing. In the genomic data outsourcing stage, multiple data owners first divide the genomic sequences locally by additive secret sharing, and then upload the shares to the two servers for constructing a joint database D. Specifically, for a sequence $S = [S[1], \ldots, S[n]]$ (n denotes the length of each sequence), a set of random numbers $r_i \in \mathbb{Z}_{2^\ell}$ are chosen, and then the data owner sets $\langle S \rangle^{\mathcal{A}} = [r_1, \ldots, r_n]$ and computes $\langle S \rangle^{\mathcal{B}} = [S[1] - r_1, \ldots, S[n] - r_n]$. After that, one share $\langle S \rangle^{\mathcal{A}}$ is uploaded to Server $\mathcal{A}$ and another share $\langle S \rangle^{\mathcal{B}}$ is uploaded to Server $\mathcal{B}$. We assume that the above shares are sent over secure channels. When genomic data outsourcing of multiple owners are completed, $\mathcal{A}$ and $\mathcal{B}$ maintain a joint genomic database D in the secret sharing form. We assume there are a total of m sequences, then $D = (\langle S_1 \rangle, \cdots, \langle S_m \rangle)$.

5.1.2 Stage 2: Query request issuing. Once the joint database D is set up, similar sequence queries can be issued from multiple users. To protect privacy, the user uses additive secret sharing to partition the query $Q = [Q[1], \cdots, Q[n]]$ into two shares denoted by $\langle Q \rangle^{\mathcal{A}} = [\langle Q[1] \rangle^{\mathcal{A}}, \cdots, \langle Q[n] \rangle^{\mathcal{A}}]$ and $\langle Q \rangle^{\mathcal{B}} = [\langle Q[1] \rangle^{\mathcal{B}}, \cdots, \langle Q[n] \rangle^{\mathcal{B}}]$, then send $\langle Q \rangle^{\mathcal{A}}$ and $\langle Q \rangle^{\mathcal{B}}$ to Server $\mathcal{A}$ and Server $\mathcal{B}$, respectively.

5.1.3 Stage 3: Secure Query Executing. When two servers receive the request from the user, secure query protocol can be executed between $\mathcal{A}$ and $\mathcal{B}$. The goal of the proposed protocol is to retrieve the indices of the top k ($k \leq m$) genomic sequences that are closest to the user query in a secure manner. The main steps involved in the secure query protocol are given in Algorithm 7.

First, we use SEED protocol (described in 4.4) directly to compute exact edit distance between $\langle Q \rangle$ and $\langle S_i \rangle$ ($1 \leq i \leq m$) without revealing any private information about the genomic database D and the query Q. Clearly, by running SEED protocol m times, a set of exact edit distance $\langle d_i \rangle = \text{SEED}(\langle Q \rangle, \langle S_i \rangle)$ ($1 \leq i \leq m$) are available. Note that $\langle d_i \rangle$ is in the secret sharing form.

Next, the two servers need to compute the k most similar genomic sequences depending on the $\langle d_i \rangle$. During this process, the values of d_i and the index i should not be revealed to the servers. To protect the confidentiality of the index, we adopt shuffle-then-compare strategy. After the secure shuffling, the link between the old sequence and the new one can be cut off. To this end, we first transform the index i to the secret sharing form. Specifically, $\mathcal{A}$ sets $\langle i \rangle^{\mathcal{A}} = i$ while $\mathcal{B}$ sets $\langle i \rangle^{\mathcal{B}} = 0$. Now that $\mathcal{A}$ and $\mathcal{B}$ hold a set of key-value pairs $[\langle i \rangle, \langle d_i \rangle](1 \leq i \leq m)$. Next we apply SSF protocol to these key-value pairs directly, i.e., the same random integers and permutation functions are applied to the shuffle of $\langle i \rangle$ and $\langle d_i \rangle$ simultaneously, for getting a set of permuted key-value pairs $[\langle \pi(\pi'(i)) \rangle, \langle d_{\pi(\pi'(i))} \rangle](1 \leq i \leq m)$. Note the security of the permuted key-value pairs is guaranteed by the randomization in SSF protocol. Similar to SMS protocol, we utilize ADD-CMP circuit to realize bubble sort for getting the top-k results (corresponding to line 10-13), denoted by $[\langle \delta_{j,0} \rangle, \langle \delta_{j,1} \rangle](1 \leq j \leq k)$. Finally,

Algorithm 7 Secure Query Protocol in SSQ-I

Input: $\mathcal{A}$ inputs $\langle Q\rangle^{\mathcal{A}}$=[$\langle Q[1]\rangle^{\mathcal{A}}, \cdots, \langle Q[n]\rangle^{\mathcal{A}}$] and $\langle S_i\rangle^{\mathcal{A}}$=[$\langle S_i[1]\rangle^{\mathcal{A}}, \cdots, \langle S_i[n]\rangle^{\mathcal{A}}$]$(1 \le i \le m)$;
$\mathcal{B}$ inputs $\langle Q\rangle^{\mathcal{B}}$=[$\langle Q[1]\rangle^{\mathcal{B}}, \cdots, \langle Q[n]\rangle^{\mathcal{B}}$] and $\langle S_i\rangle^{\mathcal{B}}$=[$\langle S_i[1]\rangle^{\mathcal{B}}, \cdots, \langle S_i[n]\rangle^{\mathcal{B}}$]$(1 \le i \le m)$
Output: $\mathcal{A}$ obtains $\langle I_j\rangle^{\mathcal{A}}$ $(1 \le j \le k)$;
$\mathcal{B}$ obtains $\langle I_j\rangle^{\mathcal{B}}$ $(1 \le j \le k)$

1: **for** $1 \le i \le m$ **do**
2: $\mathcal{A}$ and $\mathcal{B}$: $\langle d_i\rangle \leftarrow \text{SEED}(\langle Q\rangle, \langle S_i\rangle)$
3: **for** $1 \le i \le m$ **do**
4: $\mathcal{A}$: $\langle i\rangle^{\mathcal{A}} \leftarrow i$; $\mathcal{B}$: $\langle i\rangle^{\mathcal{B}} \leftarrow 0$
5: $\mathcal{A}$ and $\mathcal{B}$:
6: $L_1 \leftarrow [(\langle 1\rangle, \langle d_1\rangle), \cdots, (\langle m\rangle, \langle d_m\rangle)]$
7: $L_2 \leftarrow \text{SSF}(L_1) = [(\langle \pi(\pi'(1))\rangle, \langle d_{\pi(\pi'(1))}\rangle), \cdots, (\langle \pi(\pi'(m))\rangle, \langle d_{\pi(\pi'(m))}\rangle)]$
8: **for** $1 \le i \le m$ **do**
9: $\mathcal{A}$ and $\mathcal{B}$: $\langle \delta_{i,0}\rangle \leftarrow \langle \pi(\pi'(i))\rangle, \langle \delta_{i,1}\rangle \leftarrow \langle d_{\pi(\pi'(i))}\rangle$
10: **for** $(i = 1; i \le k; i{+}{+})$ **do**
11: **for** $(j = m; j \ge i; j{-}{-})$ **do**
12: **if** ADD-CMP$(\langle \delta_{j,1}\rangle, \langle \delta_{j-1,1}\rangle)$==0 **then**
13: $\mathcal{A}$ and $\mathcal{B}$: swap$(\langle \delta_{j,0}\rangle, \langle \delta_{j-1,0}\rangle)$, swap$(\langle \delta_{j,1}\rangle, \langle \delta_{j-1,1}\rangle)$
14: **for** $1 \le j \le k$ **do**
15: $\mathcal{A}$: $\langle I_j\rangle^{\mathcal{A}} \leftarrow \langle \delta_{j,0}\rangle^{\mathcal{A}}$
16: $\mathcal{B}$: $\langle I_j\rangle^{\mathcal{B}} \leftarrow \langle \delta_{j,0}\rangle^{\mathcal{B}}$

$\mathcal{A}$ outputs the final indices $\langle I_j\rangle^{\mathcal{A}} \leftarrow \langle \delta_{j,0}\rangle^{\mathcal{A}}$ while $\mathcal{B}$ outputs $\langle I_j\rangle^{\mathcal{B}} \leftarrow \langle \delta_{j,0}\rangle^{\mathcal{B}}$ $(1 \le j \le k)$.

5.1.4 Stage 4: Result Recovering. After $\mathcal{A}$ and $\mathcal{B}$ return $\langle I_j\rangle^{\mathcal{A}}$ and $\langle I_j\rangle^{\mathcal{B}}$ $(1 \le j \le k)$ to the query user, the user can recover the index I_j by additions locally. Note that the query user can retrieve the sequence S_{I_j} from two servers by a k-out-of-n OT protocol, which can prevent the servers from inferring the query results by monitoring the memory access.

5.2 SSQ-II (with the Approximate Edit Distance)

The above SEED protocol is an iterative process, which is hard to boost the performance. In SSQ-II, we leverage the approximate edit distance computation [3] (described in section 2.1) to improve the performance of SSQ-I. For brevity, we just present the changed parts.

5.2.1 Stage 1: Genomic data outsourcing. We assume there are ω data owners $\text{DO}_1, \cdots, \text{DO}_w$. For DO_1, given a set of genomic sequences $D_1 = (S_1, \cdots, S_{m_1})$, he/she proceeds as follows:

(1) Use *BreakToBlocks*[3] algorithm to break each sequence into t blocks. For the set D_1, DO_1 sets the sequence $S_i = (S_{i,1}, \cdots, S_{i,t}) = \textit{BreakToBlocks}(S_i)$ for all $i = 1, \cdots, m_1$.

[3]It is a partition algorithm for genomic data over plaintext, we refer the reader to [3] for details.

Algorithm 8 Secure Query Protocol in SSQ-II

Input: $\mathcal{A}$ and $\mathcal{B}$ hold D, T, L and $\langle Q\rangle$
Output: $\mathcal{A}$ obtains $\langle I_j\rangle^{\mathcal{A}}$ $(1 \le j \le k)$;
$\mathcal{B}$ obtains $\langle I_j\rangle^{\mathcal{B}}$ $(1 \le j \le k)$

1: **for** $S_i \in D_1$ **do**
2: **for** $1 \le l \le t_1, 1 \le j \le v_1$ **do**
3: $\mathcal{A}$ and $\mathcal{B}$: $\langle \chi_{l,j}\rangle \leftarrow \text{SAGSC}(\langle Q_l\rangle, \langle u_{l,j}\rangle)$
4: $\mathcal{A}$ and $\mathcal{B}$: $\langle d_i\rangle \leftarrow \sum_{l=1}^{t_1}\sum_{j=1}^{v_1} \langle \chi_{l,j}\rangle \cdot \langle \text{ED}(u_{l,j}, S_{i,l})\rangle$
5: **for** $S_i \in \{D_2, \cdots, D_\omega\}$ **do**
6: $\mathcal{A}$ and $\mathcal{B}$ do the same as line 1-4.
7: $\mathcal{A}$ and $\mathcal{B}$ hold $\langle d_i\rangle$ $(1 \le i \le m)$, the remaining steps are the same as line 3-17 in Algorithm 7.

(2) For each block location $l = 1, \cdots, t$, DO_1 gathers the set $T_1[l] = \{S_{i,l} : i = 1, \cdots, m_1\} = \{u_{l,1}, \cdots, u_{l,v_1}\}$ of all the possible sequence values for the lth block, where v_1 is the upper bound of the total number of values for each block. DO_1 pads all sets $T_1[l](1 \le l \le t)$ to the same size v_1 with dummy values. DO_1 constructs a sequence values set T_1, in which the element $T_1[l, j] = u_{l,j}(1 \le l \le t, 1 \le j \le v_1)$.
(3) For every block location $l = 1, \cdots, t$, every sequence $S_i, i = 1, \cdots, m_1$, and every value $u_{l,j} \in T_l, j = 1, \cdots, v_1$, DO_1 computes the exact edit distance between $u_{l,j}$ and $S_{i,l}$. D_1 constructs a distance set L_1, in which the element $L_1[l, j, i] = \text{ED}(u_{l,j}, S_{i,l})(1 \le l \le t, 1 \le j \le v_1, 1 \le i \le m)$.

After that, DO_1 partitions all elements in D_1, T_1, and L_1 using the genomic sequence secret sharing method shown in Section 5.1.1, and then uploads them to the servers.

$\text{DO}_2, \cdots, \text{DO}_w$ do the same as DO_1. Note that we assume all the sequences are of the same length, and the numbers of partitions for all sequences are t. After all data owners outsource their data, $\mathcal{A}$ and $\mathcal{B}$ aggregate these for maintaining a genomic database D jointly as follows:

$$D = (\langle D_1\rangle, \cdots, \langle D_w\rangle) = (\langle S_1\rangle, \cdots, \langle S_m\rangle),$$

where $m = \sum_{i=1}^{\omega} m_i$. In addition, $\mathcal{A}$ and $\mathcal{B}$ also hold

$$T = (\langle T_1\rangle, \cdots, \langle T_w\rangle), L = (\langle L_1\rangle, \cdots, \langle L_w\rangle).$$

In this stage, all computations of data owners are a one-time cost, that is, the elements in D, T, L could be reused for multiple query processes.

5.2.2 Stage 2: Query request issuing. The query user breaks the query sequence Q into t blocks to get $Q = (Q_1, \cdots, Q_t)$ by *Break-ToBlocks* algorithm, then partitions Q to obtain $\langle Q\rangle = (\langle Q_1\rangle, \cdots, \langle Q_t\rangle)$. After that, the user sends $\langle Q\rangle$ to servers.

5.2.3 Stage 3: Secure Query Executing. Recall that $\chi_{l,j}$ indicates whether or not $u_{l,j} == Q_l$. When two servers receive the request from the user, we compute the shares of the indicator bits $\langle \chi_{l,j}\rangle$ by SAGSC protocol. For $S_i \in D_1$, according to Equation 3, $\mathcal{A}$ and $\mathcal{B}$ can easily compute approximate edit distance $\langle d_i\rangle$ between Q and $S_i(1 \le i \le m_1)$ as follows:

$$\langle d_i\rangle \approx \sum_{l=1}^{t_1}\sum_{j=1}^{v_1} \langle \chi_{l,j}\rangle \cdot \langle \text{ED}(u_{l,j}, S_{i,l})\rangle,$$

where $\langle \mathrm{ED}(u_{l,j}, S_{i,l}) \rangle$ is pre-computed by the data owner and sent to the servers for store. For $S_i \in \{D_2, \cdots, D_w\}$, $\mathcal{A}$ and $\mathcal{B}$ do the same. After that, $\mathcal{A}$ and $\mathcal{B}$ hold $\langle d_i \rangle$ $(1 \le i \le m)$, the remaining steps are the same as that in SSQ-I. We describe this secure query in Algorithm 8.

5.2.4 Stage 4: Result Recovering. This stage is the same as that in SSQ-I.

6 THEORETICAL ANALYSIS

In this section, we first analyze the security of the sub-protocols and our two SSQ protocols, and then present their computational complexity.

6.1 Security Analysis

The security of secure query protocols in SSQ-I and SSQ-II is assured by the following theorems:

Theorem 6.1. *If SSF, SEED protocols, and ADD-CMP are secure under the semi-honest adversaries model, then secure query protocol in SSQ-I is secure under the semi-honest adversaries model.*

Theorem 6.2. *If SSF, SAGSC protocols and ADD-CMP are secure under the semi-honest adversaries model, then secure query protocol in SSQ-II is secure under the semi-honest adversaries model.*

In order to prove the above theorems, we first give security proofs of our sub-protocols including SSF, SBC, SMS, SEED and SAGSC protocols, and then employ composition theory [15]. The detailed proofs and discussions can be found in Appendix B.

6.2 Performance Analysis

We analyze the computational complexity of our protocols and show the analysis results in the Table 2. The detailed analysis can be found in Appendix C.

Table 2: Computational Complexity of Existing Solutions and Ours (σ is the statistical security parameter in [26])

Solutions	Enc	Dec	Mul	non-XOR gates
SSF/[26]	$2n/4n$	$2n/n$	$n/2n$	–
SBC/[26]	8/16	8/4	4/8	$3\ell/3\ell + 3\sigma + 1$
SMS/[26]	$2n/3n$	$2n/n$	$n/3n$	$3\ell(n-1)/$ $(3\ell + 3\sigma + 1)(n-1)$
SEED	$22n_1n_2$	$22n_1n_2$	$11n_1n_2$	$12\ell n_1 n_2$
SAGSC	–	–	–	$2\ell - 1$
SSQ-I	$O(mn^2)$	$O(mn^2)$	$O(mn^2)$	$O(m\ell(n^2 + k))$
SSQ-II	$O(n)$	$O(n)$	$O(n)$	$O(m\ell(tv + k))$

7 EXPERIMENTAL EVALUATION

In this section, we evaluate the performance of our protocols. As there is no full implementation for secure SSQ on outsourced genomic data so far, we conduct two sets of experiments: in the first, we compare our sub-protocols with the state-of-the-art protocols, and in the second we implement our two SSQ schemes based on our sub-protocols and report the performance of SSQ-I and SSQ-II.

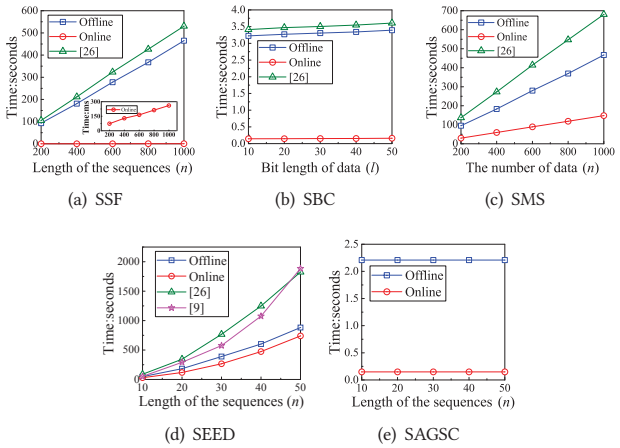


Figure 3: The performance of the protocols.

We implemented our protocols in Java, specifically, Paillier cryptosystem with distributed decryption was achieved by BigInteger Class and Yao's garbled circuits are constructed by FastGC [18], which applies the free-XOR technique [24] and oblivious-transfer extension [20] for optimizations. Our experiments are conducted on two Amazon EC2 t2.large machines (as server $\mathcal{A}$ and $\mathcal{B}$) running Ubuntu 16.04, with 8GB of RAM each, and a laptop (as the data owner and user) running Linux 16.04, with 1.70-GHz Intel i5-3317U CPU and 4 GB of memory. The communication bandwidth between the two machines for LAN setting is set to 1 GB/s and the average network latency is 0.037 ms. In our experiment, we used Homo Sapiens Mitochondrion Complete Genome dataset [30] that consists of 1,046 genomic sequences and the length of each one is 16568.

We report the results for both the offline and online time. The offline phase includes all computation and communication that are independent of the input (e.g. The generation of multiplication triplets, the construction of garbled circuits and offline part in SSF protocol), while online phase consists of all input-dependent steps.

7.1 Our Protocols vs. Previous Protocols

We compare our protocols with the ones in a previous work [26] over the real-world dataset. Recall that this work, which focuses on the secure similarity computation upon trajectory data, achieves the same functions as part of our protocols with a similar security level. For a fair comparison, we implement SSF protocol without data packing technology [6]. We also set public-key security parameter (i.e., the length of N) $\varphi = 2048$, and the symmetric security parameter $\kappa = 128$ for garbled circuits.

We first compare SSF protocol with the previous protocol in [26] for varying length of the sequences (i.e., n from 200 to 1000). As shown in Figure 3(a), the time cost in our protocol and previous protocol [26] increases linearly with the length of sequences n, which is consistent with our complexity analysis. There is a little gap between the offline time in SSF protocol and the time in previous protocol, but the online time in our protocol has remarkable advantages, as it does not involve any time-consuming cryptographic operation.

Figure 3(b) presents the time cost in SBC protocol and the previous protocol [26] as the bit length of data (ℓ) increases. The time cost

in previous protocol and ours grows linearly with ℓ, but only small increases for both. That is, the performance impact of ℓ is limited. The reason is that ℓ only affects the number of the non-XOR gates while the single-gate average computing time is very short. We can also see that the online time cost is quite little. This is because that SBC protocol mainly calls SSF protocol and ADD-CMP circuit, in which the online cost is a small portion of the overall cost. We test the impact of ℓ because it has the effect on the security of SSF protocol. To guarantee the security, our goal is to make the masked value $u_i + r_i$ (in line 4 of Algorithm 1) distribute in the interval $[0, N)$ uniformly, and the smaller ℓ would make $(N - K \cdot 2^{\ell})/N$ decreases, i.e., make the values distribute more uniformly. In general, given a fixed N, the smaller ℓ is, the stronger the security is. Because of this, we set a smaller $\ell = 10$ in all other tests.

Figure 3(c) shows the time cost in SMS protocol and the previous protocol [26] increases at a linear speed as the number of data (n) increases. For the same reasons explained above, SMS protocol can also be divided into two phases and the online time is much less than that in the previous work.

We compare our SEED protocol with the advanced protocol in [9]. This work presents a homomorphic computation of exact edit distance based on somewhat homomorphic encryption scheme. In addition, we also implement another secure exact edit distance computing protocol using the protocols in [26] as a comparison. Figure 3(d) shows the time cost for these protocols. When the length of the sequences n ($n_1 = n_2 = n$) increases, the cost of all protocols grows as $O(n^2)$, since exact edit distance computing is a two-dimension iteration algorithm. Even so, the performance of SEED protocol still has advantages over the other two. Specially, SEED has about 50% performance improvement compared with the protocol in [9]. For example, when $n = 50$, the online time is 740 seconds in SEED protocol and 1885 seconds in [9].

For SAGSC protocol, Figure 3(e) shows that the online time is far less than the offline time, since the major operation in SAGSC is the computation of EQ-ADD circuit and a single circuit runtime is much less than the circuit construction time. We also observe the impact of the sequence length (n) to SAGSC protocol is limited. It is because that the increase of n only increases the number of modular addition operations, which has a very small impact on the online time compared with time cost of EQ-ADD circuit computation.

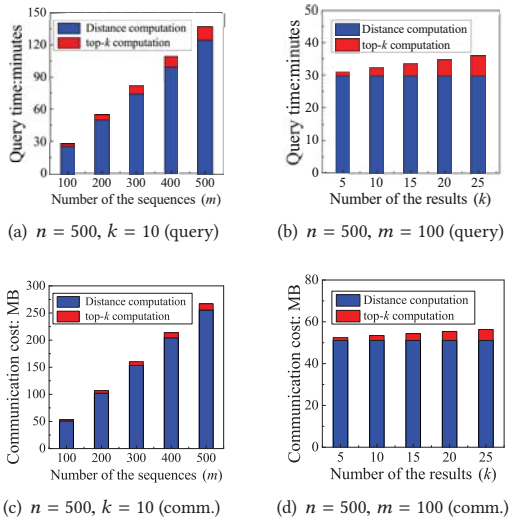
7.2 The Performance of SSQ-I and SSQ-II

In our two schemes, the query executing stage includes the edit distance computation and top-k results retrieve. We use the SEED protocol to compute the exact edit distance in SSQ-I. In SSQ-II, an approximate edit distance computation method is employed for efficiency. We customize genomic sequences of different length (n from 10 to 50) based on the real-world data as test dataset and set block number t from 2 to 10 in SSQ-II, then report single edit distance computation time for two schemes in Table 3. We only show the online time in this subsection. Clearly, if the length of genomic sequences is large, SEED used by SSQ-I generates a large amount of computing overhead (e.g., it takes 740.3s when $n = 50$).

Next, we evaluate the performance of SSQ-II in the online phase. We assume there are 500 genomic sequences with length as 500, which are outsourced by two data owners and the data size of each

Table 3: The edit distance computation time in SSQ-I and SSQ-II

n	10	20	30	40	50
SSQ-I	29.6s	118.4s	266.4s	473.6s	740.3s
SSQ-II	1.2s	2.2s	3.4s	4.7s	6.0s



(a) $n = 500$, $k = 10$ (query) (b) $n = 500$, $m = 100$ (query)

(c) $n = 500$, $k = 10$ (comm.) (d) $n = 500$, $m = 100$ (comm.)

Figure 4: The performance of SSQ-II protocol.

owner is the same. We test SSQ-II over this dataset with different scales (m from 100 to 500) while fixing the number of blocks in each sequence $t = 20$. Figure 4(a) shows the query time for SSQ-II as m increases (while fixing $k = 10$ and k is the number of the results). The query time grows linearly with m, which coincides with the theoretical analysis in Section 6.2. We can also conclude that the time for computing edit distance makes up a greater share of the total time compared with the time for top-k results retrieve. For example, it takes 124 minutes for distance computation and 12.5 minutes for top-k computation when $m = 500$. Figure 4(b) shows the time cost in SSQ-II increases as k increases (while fixing $m = 100$), since there is a positive correlation between the time for top-k computation and the value of k, although k has no effect on the time for distance computation. We also collect communication cost between the two servers during the above tests, similar varying tendencies are shown in Figure 4(c) and 4(d). The underlying reason is that the increase of sequence length (m) and the result number (k) will undoubtedly raise the cost of communication. However, the communication cost between the client and the servers is negligible, which is one round including only one query request and k result indices.

During the query stage, data owners and users needn't participate in any computation. During the outsourcing stage, data owners perform all of the operations in plaintext forms. The data owner only takes about 3 minutes to create a database in which $m = 500$, $n = 500$ and $t = 20$. For arbitrary query users, the task is just partitioning queries and recovering the secret sharing results by modular addition operations. This means the overhead to users is lightweight, which further makes our schemes scalable.

Table 4: Accuracy of SSQ-II for various choice of k and t when $m = 500$ and $n = 500$

Number of blocks	Number of results	Accuracy
$t = 10$	$k = 1/5/10$	80%/76%/76%
$t = 15$	$k = 1/5/10$	90%/84%/87%
$t = 20$	$k = 1/5/10$	100%/94%/92%
$t = 25$	$k = 1/5/10$	100%/98%/95%
$t = 30$	$k = 1/5/10$	100%/100%/96%

Recall from Section 5.2 that SSQ-II yields approximate results, computing edit distance through partitioning method and SAGSC protocol could introduce error. To assess the effects of SAGSC protocol, we run this protocol multiple times and count the number of false-positive results. As the SAGSC protocol is used to compare the sequences in the same location from the sequence values set T_i with the query block Q_i, we collect all sequence values in T_i from above experiments and then launch pairwise comparison for these sequences in the same location by SAGSC protocol. We made a total of 486 comparisons and the number of false-positive results is 4, the error rate is only about 0.82%. As for the error caused by partitioning method, we refer the reader to [3] for details.

The accuracy results of SSQ-II are summarized in Table 4. Note that we do the tests over the above real-world dataset ($m = 500, n = 500$) and repeat the experiment 10 times for random choice of the queries. From the table, we find that when the number of blocks is no fewer than 20, the accuracies of all cases ($k = 1, 5, 10$) are over 90%. Consider the query performance and accuracy together, so we set $t = 20$ when $m = 500, n = 500$ in the above experiments.

8 DISCUSSION

8.1 Reducing Error Rate

Here we theoretically analyze how to further reduce the error rate of SAGSC protocol by a well-designed encoding scheme. Consider two genomic sequences in which at least one nucleotide's amount is different, and for each nucleotide, when the number of its positions at which the corresponding symbols are different is below a (a is a integer), we can deterministically detect if the two sequences are different by an encoding scheme in a base. For example, "ATAGCG" and "CTACCG" are represented as $x = [0, 3, 0, 2, 1, 2]$ and $y = [1, 3, 0, 1, 1, 2]$ under existing encoding scheme, the equation $\sum_{i=1}^{6} x_i == \sum_{i=1}^{6} y_i$ holds, it makes a false-positive result when using SAGSC protocol. If we use a decimal encoding scheme (i.e., a=10, we use 0, 10, 100, 1000 to represent four nucleotides A,C,G,T), the same sequences can be denoted as $x = [0, 1000, 0, 100, 10, 100]$ and $y = [10, 1000, 0, 10, 10, 100]$, and the equation $\sum_{i=1}^{6} x_i == \sum_{i=1}^{6} y_i$ does not hold, i.e., we can detect the two sequences are different.

8.2 Improving Query Efficiency

In the process of query, we compute edit distances in multiple datasets uploaded by different data owners. In fact, these datasets may contain duplicate records, we can decrease the size of genomic data by deduplication, so the query algorithm is in high efficiency. In addition, the online query cost can be greatly reduced by parallel computing, the reason is that the computations of $\langle \chi_{l,j} \rangle$ and $\langle d_i \rangle$ (i.e., Stage 3 in SSQ-II) are independent of others. We consider the multi-source genomic data fusion and the parallelization of the query program as a future work.

9 RELATED WORK

Privacy-preserving query and analysis over genetic data have received much attention recently. Existing works [7, 11, 16, 33, 34, 37] deal with genome variant query, pattern matching, range query, count query and statistic information computation over genomic data in a privacy-preserving manner. Secure similar sequence query on genomic data, which is the focus of our research, is a special case of secure query processing on genomic data. Secure SSQ on genetic data is usually considered under two different system models. The first model is secure multi-party computation setting. The second model is the secure outsourcing of computation model. We review the related works under each system model respectively.

9.1 SSQ in Multi-party Computation Model

Jha et al. proposed a privacy-preserving protocol to compute the edit distance between two genomic sequences based on dynamic programming [22]. For performance reasons, the authors showed three methods to replicate the original edit distance algorithm over Garbled Circuits. Further, Zhu et al. [38] customized a secure garbling scheme to compute edit distance by leveraging publicly exploitable traits of target computations, which is significantly more efficient than traditional garbled circuits. Yet, the computation and the communication overhead of the above schemes are considerable, since these protocols are all iterative and the number of iterations is the product of the lengths of two input sequences. To exploit the high similarity between human genomic sequences, Wang et al. [36] resorted to the approximate edit distance computing. Their scheme is efficient as the edit distance computation problem is transformed to the relatively simple set-intersection-size-approximation problem. Inspired by this approach, a series of query protocols [2, 3] based on approximate edit distance are proposed. In the most advanced solution [3], the authors partitioned the sequences into smaller blocks and then pre-computed the edit distance within the blocks. In this way, a lot of computational overhead is transferred to a preprocessing stage. These works, however, are all in the SMC model (data is distributed between two parties who collaboratively compute the results without revealing to each other their private data), which are not suitable to our problem since we aim at reducing user-side overhead to the minimum.

9.2 SSQ in Outsourcing Computation Model

So far there has been no full implementation for secure SSQ on outsourced genomic data. The following related works focused on edit distance computation. Atallah and Li [5] initially studied the secure outsourced protocol for edit-distance sequence comparisons. Emiliano et al. [10] and Cheon et al. [9] discussed how to calculate edit distance based on homomorphic encryption (HE). However, in the above schemes, the exact edit distance is computed in a recursive way, so computational overhead and communication costs are too large. For example, the scheme in [9] takes 27 seconds to compute an 8×8 block by dynamic programming. A further work in [23] presented a secure approximate edit distance protocol based

on the comparison binary circuit in [9]. However, the query user has a massive overhead. A recent work [27] took a step back to study secure similar genomic sequence query based on Hamming distance instead of edit distance. But this simple indicator of similarity is not suitable for genomic data as single-gene insertion or deletion would have much impact on the similarity. Another similar work [26] focuses on the similarity computation on encrypted trajectories data. Compared with it, our mixed protocols achieve higher performance and support a richer retrieve function. More importantly, all above works are not scalable in supporting multiple users and data owners, which is a big gap that our work aims to fill.

10 CONCLUSION

We have presented two novel schemes SSQ-I and SSQ-II towards secure similar-sequence-query on outsourced genomic data. At the core of our schemes, we have designed a set of mixed secure protocols based on secret sharing, garbled circuit, and partial homomorphic encryptions to achieve security, efficiency, and scalability simultaneously. Formal security analysis demonstrated all protocols are secure under the semi-honest adversary model. Finally, we show the efficacy and efficiency of our solutions through extensive experiments conducted over a real genomic dataset on a commercial cloud platform. For the future work, we will continue to improve SSQ performance over a larger dataset.

ACKNOWLEDGMENTS

We thank the shepherd and reviewers for the valuable suggestions. This work is funded in part by National Science Foundation of China under number U1736216.

REFERENCES

[1] 2017. Illumina wants to sequence your whole genome for 100. (2017). https://techcrunch.com/2017/01/10/illumina-wants-to-sequence-your-whole-genome-for-100.

[2] Md Momin Al Aziz, Dima Alhadidi, and Noman Mohammed. 2017. Secure approximation of edit distance on genomic data. *BMC medical genomics* 10, 2 (2017), 41.

[3] Gilad Asharov, Shai Halevi, Yehuda Lindell, and Tal Rabin. 2017. Privacy-Preserving Search of Similar Patients in Genomic Data. *IACR Cryptology ePrint Archive* 2017 (2017), 144.

[4] Mikhail Atallah, Marina Bykova, Jiangtao Li, Keith Frikken, and Mercan Topkara. 2004. Private collaborative forecasting and benchmarking. In *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*. ACM, 103–114.

[5] Mikhail J Atallah and Jiangtao Li. 2005. Secure outsourcing of sequence comparisons. *International Journal of Information Security* 4, 4 (2005), 277–287.

[6] Tiziano Bianchi, Alessandro Piva, and Mauro Barni. 2010. Composite signal representation for fast and storage-efficient processing of encrypted signals. *IEEE Transactions on Information Forensics and Security* 5, 1 (2010), 180–187.

[7] Feng Chen, Chenghong Wang, Wenrui Dai, Xiaoqian Jiang, Noman Mohammed, Md Momin Al Aziz, Md Nazmus Sadat, Cenk Sahinalp, Kristin Lauter, and Shuang Wang. 2017. PRESAGE: PRivacy-preserving gEnetic testing via SoftwAre Guard Extension. *BMC medical genomics* 10, 2 (2017), 48.

[8] Ke Cheng, Yantian Hou, and Liangmin Wang. 2018. Secure Similar Sequence Query on Outsourced Genomic Data. (2018). http://cs.boisestate.edu/~yhou/gene_search_tech_report.pdf

[9] Jung Hee Cheon, Miran Kim, and Kristin Lauter. 2015. Homomorphic computation of edit distance. In *International Conference on Financial Cryptography and Data Security*. Springer, 194–212.

[10] Emiliano De Cristofaro, Kaitai Liang, and Yuruo Zhang. 2016. Privacy-Preserving Genetic Relatedness Test. *arXiv preprint arXiv:1611.03006* (2016).

[11] Daniel Demmler, Kay Hamacher, Thomas Schneider, and Sebastian Stammler. 2017. Privacy-Preserving Whole-Genome Variant Queries. In *16. International Conference on Cryptology And Network Security (CANS'17) (LNCS)*. Springer. http://thomaschneider.de/papers/DHSS17.pdf To appear.

[12] Daniel Demmler, Thomas Schneider, and Michael Zohner. 2015. ABY-A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *NDSS*.

[13] Yousef Elmehdwi, Bharath K Samanthula, and Wei Jiang. 2014. Secure k-nearest neighbor query over encrypted data in outsourced environments. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*. IEEE, 664–675.

[14] De Cristofaro Emiliano, Liang Kaitai, and Zhang Yuruo. 2016. Privacy-Preserving Genetic Relatedness Test. In *GenoPri'16: 3rd International Workshop on Genome Privacy and Security*. Chicago, IL, USA.

[15] Oded Goldreich. 2009. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press.

[16] Mohammad Zahidul Hasan, Md Safiur Rahman Mahdi, and Noman Mohammed. 2017. Secure Count Query on Encrypted Genomic Data. *arXiv preprint arXiv:1703.01534* (2017).

[17] Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, and Tomas Toft. 2012. Efficient RSA Key Generation and Threshold Paillier in the Two-Party Setting. In *CT-RSA*. Springer, 313–331.

[18] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. 2011. Faster Secure Two-Party Computation Using Garbled Circuits.. In *USENIX Security Symposium*, Vol. 201.

[19] Mathias Humbert, Erman Ayday, Jean-Pierre Hubaux, and Amalio Telenti. 2013. Addressing the concerns of the lacks family: quantification of kin genomic privacy. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 1141–1152.

[20] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. 2003. Extending Oblivious Transfers Efficiently.. In *Crypto*, Vol. 2729. Springer, 145–161.

[21] Neda Jahanshad, Priya Rajagopalan, Xue Hua, Derrek P Hibar, Talia M Nir, Arthur W Toga, Clifford R Jack, Andrew J Saykin, Robert C Green, Michael W Weiner, et al. 2013. Genome-wide scan of healthy human connectome discovers SPON1 gene variant influencing dementia severity. *Proceedings of the National Academy of Sciences* 110, 12 (2013), 4768–4773.

[22] Somesh Jha, Louis Kruger, and Vitaly Shmatikov. 2008. Towards practical privacy for genomic computation. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*. IEEE, 216–230.

[23] Miran Kim and Kristin Lauter. 2015. Private genome analysis through homomorphic encryption. *BMC medical informatics and decision making* 15, 5 (2015).

[24] Vladimir Kolesnikov and Thomas Schneider. 2008. Improved garbled circuit: Free XOR gates and applications. *Automata, Languages and Programming* (2008), 486–498.

[25] Yehuda Lindell and Benny Pinkas. 2009. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology* 22, 2 (2009), 161–188.

[26] An Liu, Kai Zhengy, Lu Liz, Guanfeng Liu, Lei Zhao, and Xiaofang Zhou. 2015. Efficient secure similarity computation on encrypted trajectory data. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*. IEEE, 66–77.

[27] Md Safiur Rahman Mahdi, Mohammad Zahidul Hasan, and Noman Mohammed. 2017. Secure Sequence Similarity Search on Encrypted Genomic Data. In *Connected Health: Applications, Systems and Engineering Technologies (CHASE), 2017 IEEE/ACM International Conference on*. IEEE, 205–213.

[28] P. Mohassel and Y. Zhang. 2017. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *2017 IEEE Symposium on Security and Privacy (SP)*. 19–38. https://doi.org/10.1109/SP.2017.12

[29] Cancer Genome Atlas Network et al. 2012. Comprehensive molecular portraits of human breast tumours. *Nature* 490, 7418 (2012), 61–70.

[30] Anna Olivieri, Carlo Sidore, and et al. 2017. Mitogenome diversity in Sardinians: a genetic window onto an island's past. *Molecular biology and evolution* 34, 5 (2017), 1230–1239.

[31] Pascal Paillier et al. 1999. Public-key cryptosystems based on composite degree residuosity classes. In *Eurocrypt*, Vol. 99. Springer, 223–238.

[32] Suyash S Shringarpure and Carlos D Bustamante. 2015. Privacy risks from genomic data-sharing beacons. *The American Journal of Human Genetics* 97, 5 (2015), 631–646.

[33] Wenhai Sun, Ning Zhang, Wenjing Lou, and Y Thomas Hou. 2017. When gene meets cloud: Enabling scalable and efficient range query on encrypted genomic data. In *INFOCOM 2017*. IEEE, 1–9.

[34] Bing Wang, Wei Song, Wenjing Lou, and Y Thomas Hou. 2017. Privacy-preserving pattern matching over encrypted genetic data in cloud computing. In *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*. IEEE, 1–9.

[35] Shuang Wang, Xiaoqian Jiang, Haixu Tang, Xiaofeng Wang, Diyue Bu, Knox Carey, Stephanie OM Dyke, Dov Fox, Chao Jiang, Kristin Lauter, et al. 2017. A community effort to protect genomic data sharing, collaboration and outsourcing. *npj Genomic Medicine* 2, 1 (2017), 33.

[36] Xiao Shaun Wang, Yan Huang, Yongan Zhao, Haixu Tang, XiaoFeng Wang, and Diyue Bu. 2015. Efficient genome-wide, privacy-preserving similar patient query based on private edit distance. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 492–503.

[37] Jun Zhou, Zhenfu Cao, and Xiaolei Dong. 2016. PPOPM: more efficient privacy preserving outsourced pattern matching. In *European Symposium on Research in Computer Security*. Springer, 135–153.

[38] Ruiyu Zhu and Yan Huang. 2017. *Efficient privacy-preserving general edit distance and beyond*. Technical Report. Cryptology ePrint Archive, Report 2017/683, 2017. http://eprint. iacr. org/2017/683 10 April 2017, date last accessed.

A WAGNER-FISHER ALGORITHM

Algorithm 9 shows how to compute edit distance by Wagner-Fisher algorithm. The edit distance between two strings $X = [a_1, a_2, \ldots, a_{n_1}]$ and $Y = [b_1, b_2, \ldots, b_{n_2}]$ is given by d_{n_1,n_2}, calculated by the following recurrence:

$$d_{i,j} = \begin{cases} 0 & \text{if } i == 0 \text{ or } j == 0 \\ \min \begin{cases} d_{i-1,j} + c_{del} \\ d_{i,j-1} + c_{ins} \\ d_{i-1,j-1} + c_{sub} \end{cases} & \text{otherwise} \end{cases} \quad (4)$$

where $c_{ins}, c_{del}, c_{sub}$ correspond to the cost of single-character insert, delete, and substitute. For the edit distance, $c_{ins} \leftarrow 1, c_{del} \leftarrow 1, c_{sub} \leftarrow (a_i == b_j)?0 : 1$. It is clear from Equation 4 that the edit distance calculation is transformed into the process of filling an $(n_1 + 1) \times (n_2 + 1)$ table with each element $d_{i,j}$. The full dynamic programming table can be constructed in $O(n_1 n_2)$ time. Once the table is filled, it is easy to trace on the optimal path. That is, the index in Algorithm 9 reveals optimal conversion path from an original sequence to a new one. Figure 5 shows an example of edit distance computation. The lower-right element $d_{4,5}$=2 is the final result and the dotted line represents one of the optimal paths.

		C	T	A	G	A
	0	1	2	3	4	5
A	1	1	2	2	3	4
T	2	2	1	2	3	4
G	3	3	2	2	2	3
A	4	4	3	2	3	**2**

Figure 5: Example of edit distance computation (The lower-right element $d_{4,5} = 2$ is the final result and the dotted line represents one of the optimal paths).

Algorithm 9 Edit Distance Computation

Input: $X = \{a_1, a_2, \ldots, a_{n_1}\}$ and $Y = \{b_1, b_2, \ldots, b_{n_2}\}$
Output: d_{n_1,n_2}

1: **for** $0 \le i \le n_1$ **do**
2: $\quad d_{i,0} \leftarrow i$.
3: **for** $0 \le j \le n_2$ **do**
4: $\quad d_{0,j} \leftarrow j$.
5: $c_{del} \leftarrow 1, c_{ins} \leftarrow 1$.
6: **for** $1 \le i \le n_1, 1 \le j \le n_2$ **do**
7: $\quad c_{sub} \leftarrow (a_i == b_j)?0 : 1$.
8: $\quad d_{i,j} = \min(d_{i-1,j} + c_{del}, d_{i,j-1} + c_{ins}, d_{i-1,j-1} + c_{sub})$.
9: **return** d_{n_1,n_2}

B THE SECURITY ANALYSIS

Suppose that a two-party protocol P asks $\mathcal{A}$ to compute the function $f_{\mathcal{A}}(x, y)$, and asks $\mathcal{B}$ to compute $f_{\mathcal{B}}(x, y)$, where x, y are the inputs of $\mathcal{A}$ and $\mathcal{B}$, respectively. The view of $\mathcal{A}$ (resp. $\mathcal{B}$) during an execution of P on (x, y), denoted $view_{\mathcal{A}}(x, y)$ (resp. $view_{\mathcal{B}}(x, y)$), is $(x, r_{\mathcal{A}}, m_1, \cdots, m_t)$ (resp. $(y, r_{\mathcal{B}}, m_1, \cdots, m_t)$), where $r_{\mathcal{A}}$ (resp. $r_{\mathcal{B}}$) represents randomness of $\mathcal{A}$ (resp. $\mathcal{B}$) and m_i represents the i-th message passed between the parties. Also let $O_{\mathcal{A}}(x, y)$ and $O_{\mathcal{B}}(x, y)$ denote $\mathcal{A}$'s (resp. $\mathcal{B}$'s) output. We say that protocol P is secure against semi-honest adversaries if there exist probabilistic polynomial time (PPT) simulators S_1 and S_2 such that:

$$(S_1(x, f_{\mathcal{A}}(x, y)), f(x, y)) \stackrel{c}{\equiv} (view_{\mathcal{A}}(x, y), O(x, y)) \quad (5)$$

$$(S_2(y, f_{\mathcal{B}}(x, y)), f(x, y)) \stackrel{c}{\equiv} (view_{\mathcal{B}}(x, y), O(x, y)) \quad (6)$$

where $\stackrel{c}{\equiv}$ denotes computational indistinguishability. More details can be found in [15].

We assume that the secure computation primitives (described in Section 2.2) involved in our protocols are secure under semi-honest model [15], and Paillier cryptosystem with distributed decryption is semantically secure. The formal security proofs of them can be found in [4, 17]. In addition, we note that ADD-CMP and EQ-ADD are direct applications of Yao's garbled circuits, whose security proof can be found in [25]. Next, we prove the following theorem under the above security assumptions.

B.1 Security Analysis of Sub-protocols

We first prove the security of all our sub-protocols. Then we will employ composition theory to prove that our SSQ-I and SSQ-II are secure.

Theorem B.1. *If Paillier cryptosystem with distributed decryption (short as PCDD) [17] is semantically secure, then the offline phase of secure shuffling (short as SSF_off) protocol is secure under the semi-honest adversaries model.*

Proof. The correctness is easy to see, just by inspecting the output result is shuffled from $[-u_1 - v_1, \cdots, -u_n - v_n]$ by the permutation function $\pi(\pi'(\cdot))$. Recall that the function of *SSF_off* is that $\mathcal{A}$ selects a set of random integers $u_1, \cdots, u_n$, $\mathcal{B}$ selects a set of random integers $v_1, \cdots, v_n$ and then $\mathcal{B}$ outputs a set of shuffled values $[-u_{\pi(\pi'(1))} - v_{\pi(\pi'(1))}, \ldots, -u_{\pi(\pi'(n))} - v_{\pi(\pi'(n))}]$. As for security, we construct simulators in two distinct cases as *SSF_off* protocol is asymmetric for two parties. Case 1: $\mathcal{A}$ is corrupted by an adversary. We construct a simulator S_1 to simulate $\mathcal{A}$'s view. For $\mathcal{A}$ receives $L_1 = [E_{pk}(u_{\pi'(1)} + v_{\pi'(1)} + r_{\pi'(1)}), \ldots, E_{pk}(u_{\pi'(n)} + v_{\pi'(n)} + r_{\pi'(n)})]$ from $\mathcal{B}$ in the real world, S_1 randomly picks a set of integers $\alpha_1, \cdots, \alpha_n$ from $\mathbb{Z}_N$ and encrypts them by PCDD to get $L'_1 = [E_{pk}(\alpha_1), \cdots, E_{pk}(\alpha_n)]$. Then, any PPT adversary cannot distinguish the simulator's encryption of the random integers (L'_1) from $\mathcal{B}$'s encryption of the correct computation (L_1) due to the semantical security of PCDD. For the output of this protocol in the real world $[-u_{\pi(\pi'(1))} - v_{\pi(\pi'(1))}, \ldots, -u_{\pi(\pi'(n))} - v_{\pi(\pi'(n))}]$, it is clearly computational indistinguishable from the output of *SSF_off*' function. In addition, all the values in L_1 and the real output are irrelevant, and all the values in L'_1 and the function output are also irrelevant, so these values are computational indistinguishable. Therefore, Equation 5 clearly holds. Case 2: $\mathcal{B}$ is corrupted by an adversary. We construct a simulator S_2 to simulate the message sent by $\mathcal{A}$. For $\mathcal{B}$ receives $L_0 = [E_{pk}(u_1 + r_1), \ldots, E_{pk}(u_n + r_n)]$ from $\mathcal{A}$, S_2 randomly picks a set of integers $\beta_1, \cdots, \beta_n$ from $\mathbb{Z}_N$ and encrypts them by PCDD to get $L'_0 = [E_{pk}(\beta_1), \cdots, E_{pk}(\beta_n)]$. For $\mathcal{B}$ receives L_2 from $\mathcal{A}$, S_2 randomly picks a set of integers $\gamma_1, \cdots, \gamma_n$ from $\mathbb{Z}_N$, then encrypts them by PCDD to get L'_2. Any PPT adversary who can distinguish between interaction with $\mathcal{A}$

(i.e., L_0, L_2) and interaction with S_2 (i.e., L_0', L_2') can be used to break the semantical security of PCDD. In addition, all the above values are selected independently. Thus, no such adversary exists, which means Equation 6 holds. Putting the above results together, we can claim that *SSF_off* is secure under the semi-honest adversary model. □

Theorem B.2. *The online phase of secure shuffling (short as SSF_on) protocol is secure under the semi-honest adversaries model.*

Proof. The proof of Theorem B.2 is similar to that of Theorem B.1. Due to space limitation, we only describe the outline of our proof: the function of *SSF_on* is that $\mathcal{A}$ and $\mathcal{B}$ input a secret sharing sequence $\langle \mathbf{x} \rangle$, then $\mathcal{A}$ outputs a share of the shuffled sequence $\langle \mathbf{x}' \rangle$. The exchanged messages $L_4 = [\langle x_1 \rangle^{\mathcal{A}} + u_1, \ldots, \langle x_n \rangle^{\mathcal{A}} + u_n]$ can be simulated by randomly choosing $L_4' = [\beta_1, \cdots, \beta_n]$ from $\mathbb{Z}_{2^\ell}$ and $L_5 = [x_{\pi'(1)} + u_{\pi'(1)} + v_{\pi'(1)}, \ldots, x_{\pi'(n)} + u_{\pi'(n)} + v_{\pi'(n)}]$ can be simulated by randomly choosing $L_5' = [r_1, \cdots, r_n]$ from $\mathbb{Z}_{2^\ell}$. So we can claim that *SSF_on* is secure under the semi-honest adversaries model. Please see the technical report [8] for the detailed proof. □

Theorem B.3. *If SSF protocol and ADD-CMP are secure under the semi-honest adversaries model, then SBC protocol is secure under the semi-honest adversaries model.*

Proof. The correctness is easy to see by inspecting the output $\langle y \rangle$ is actually assigned according to the relationship between x_1 and x_2. As for security, we present the security proof in a hybrid model [15] where $\mathcal{A}$ and $\mathcal{B}$ have access to a trusted party (TP) which can realize the function of SSF protocol and ADD-CMP. $\mathcal{A}$ and $\mathcal{B}$ call TP to run the function of SSF protocol with input $[\langle x_1 \rangle, \langle x_2 \rangle]$ for outputting the random permutation result $[\langle x_{\pi(\pi'(1))} \rangle^{\mathcal{A}}, \langle x_{\pi(\pi'(2))} \rangle^{\mathcal{A}}]$ to $\mathcal{A}$ and $[\langle x_{\pi(\pi'(1))} \rangle^{\mathcal{B}}, \langle x_{\pi(\pi'(2))} \rangle^{\mathcal{B}}]$ to $\mathcal{B}$. These values are all random according to the function of SSF protocol, therefore, in the case of a corrupted $\mathcal{A}$, the simulator S_1 just chooses two integers α_1, α_2 from $\mathbb{Z}_{2^\ell}$ uniformly at random to simulate $\langle x_{\pi(\pi'(1))} \rangle^{\mathcal{A}}, \langle x_{\pi(\pi'(2))} \rangle^{\mathcal{A}}$, and in the case of a corrupted $\mathcal{B}$, the simulator S_2 just do the same as S_1 to simulate $\langle x_{\pi(\pi'(1))} \rangle^{\mathcal{B}}$, $\langle x_{\pi(\pi'(2))} \rangle^{\mathcal{B}}$ uniformly at random. $\mathcal{A}$ and $\mathcal{B}$ call TP to run the function of SSF protocol with input $[\langle y_1 \rangle, \langle y_2 \rangle]$ for outputting the random permutation result $[\langle y_{\pi(\pi'(1))} \rangle^{\mathcal{A}}, \langle y_{\pi(\pi'(2))} \rangle^{\mathcal{A}}]$ to $\mathcal{A}$ and $[\langle y_{\pi(\pi'(1))} \rangle^{\mathcal{B}}, \langle y_{\pi(\pi'(2))} \rangle^{\mathcal{B}}]$ to $\mathcal{B}$. Clearly, S_1 and S_2 can do the same as above for simulation. $\mathcal{A}$ and $\mathcal{B}$ call TP to run the function of ADD-CMP with input $\langle x_{\pi(\pi'(1))} \rangle, \langle x_{\pi(\pi'(2))} \rangle$ for outputting θ to $\mathcal{A}$ and $\mathcal{B}$. The case in which θ is assigned to 0 or 1 is randomly given, as $\pi(\pi'(\cdot))$ is randomly selected. Thus, in the case of a corrupted $\mathcal{A}$ or a corrupted $\mathcal{B}$, the simulator just chooses a random integer from $\{0, 1\}$ to simulate θ. According to the composition theory [15], combining the above analysis and the security of SSF protocol and ADD-CMP, we can claim that SBC protocol is secure under the semi-honest adversaries model. □

Theorem B.4. *If SSF protocol and ADD-CMP are secure under the semi-honest adversaries model, then SMS protocol is secure under the semi-honest adversaries model.*

Proof. The proof of Theorem B.4 is similar to that of Theorem B.3. Due to space limitation, we only describe the outline of our proof. We assume $\mathcal{A}$ and $\mathcal{B}$ have access to a trusted party (TP) which can realize the function of SSF protocol and ADD-CMP. The exchanged messages $\langle x_{\pi(\pi'(i))} \rangle^{\mathcal{A}}$ $(1 \le i \le n)$ can be simulated by randomly choosing the values $\alpha_i (1 \le i \le n)$ at random from $\mathbb{Z}_{2^\ell}$ and θ can be simulated by choosing a random integer from $\{0, 1\}$. So we can claim that SMS protocol is secure under the semi-honest adversaries model. Please see the technical report [8] for the detailed proof. □

Theorem B.5. *If SBC and SMS protocols are secure under the semi-honest adversaries model, then SEED protocol is secure under the semi-honest adversaries model.*

Proof. The correctness is easy to see by recover d_{ED} from the output $\langle d_{ED} \rangle$ and then inspecting it is actually equal to the edit distance between $\mathbf{x}$ and $\mathbf{y}$. As for security, we present the security proof in a hybrid model where $\mathcal{A}$ and $\mathcal{B}$ have access to a trusted party (TP) which can realize the function of SBC and SMS protocols. $\mathcal{A}$ and $\mathcal{B}$ call TP to run the function of SBC with input $\langle t_1 \rangle, \langle x_i \rangle, \langle z_0 \rangle, \langle z_1 \rangle$ for outputting $\langle t_3 \rangle^{\mathcal{A}}$ to $\mathcal{A}$ and $\langle t_3 \rangle^{\mathcal{B}}$ to $\mathcal{B}$. $\langle t_3 \rangle^{\mathcal{A}}$ and $\langle t_3 \rangle^{\mathcal{B}}$ are both random according to the function of SBC protocol (the output is in the secret sharing form), therefore, in the case of a corrupted $\mathcal{A}$, the simulator S_1 just chooses a integers α from $\mathbb{Z}_{2^l}$ uniformly at random to simulate $\langle t_3 \rangle^{\mathcal{A}}$. In the case of a corrupted $\mathcal{B}$, the simulator S_2 do the same as S_1 to simulate $\langle t_3 \rangle^{\mathcal{B}}$. $\mathcal{A}$ and $\mathcal{B}$ call TP to run the function of SBC with input $\langle x_i \rangle, \langle t_2 \rangle, \langle t_3 \rangle, \langle z_1 \rangle$. Clearly, S_1 and S_2 can do the same as above for simulation. $\mathcal{A}$ and $\mathcal{B}$ call TP to run the function of SMS, the outputs of this function $\langle d_{i,j} \rangle^{\mathcal{A}}$ and $\langle d_{i,j} \rangle^{\mathcal{B}}$ are both random according to the function of SMS protocol (the output is in the secret-sharing form), therefore, in the case of a corrupted $\mathcal{A}$ or a corrupted $\mathcal{B}$, S_1 or S_2 can do the same as above for simulation. According to the composition theory, combining the above analysis and the security of SBC and SMS protocols, we can claim that SEED protocol is secure under the semi-honest adversaries model. □

Theorem B.6. *If EQ-ADD is secure under the semi-honest adversaries model, then SAGSC protocol is secure under the semi-honest adversaries model.*

Proof. The proof of Theorem B.6 is similar to that of Theorem B.3. Due to space limitation, we only describe the outline of our proof. We assume $\mathcal{A}$ and $\mathcal{B}$ have access to a trusted party (TP) which can realize the function of EQ-ADD. The messages $\langle \chi \rangle^{\mathcal{B}}$ can be simulated by choosing a random integer from $\mathbb{Z}_{2^\ell}$. So we can claim that SAGSC protocol is secure under the semi-honest adversaries model. Please see the technical report [8] for the detailed proof. □

B.2 Security Analysis of SSQ-I and SSQ-II

Now, we prove Theorem 6.1 and Theorem 6.2 based on the above theorems.

Proof. (Theorem 6.1) The correctness is easy to see by recover I_j from the output $\langle I_j \rangle$ and then inspecting it does actually belong to the index set of top-k results closed to query. As for security, we present the security proof in a hybrid model where $\mathcal{A}$ and $\mathcal{B}$ have access to a trusted party (TP) which can realize the function of SSF, SEED protocols and ADD-CMP. $\mathcal{A}$ and $\mathcal{B}$ call TP to run the

function of SEED with input $\langle Q \rangle$ and $\langle S_i \rangle$ for outputting $\langle d_i \rangle^{\mathcal{A}}$ to $\mathcal{A}$ and $\langle d_i \rangle^{\mathcal{B}}$ to $\mathcal{B}$. $\langle d_i \rangle^{\mathcal{A}}$ and $\langle d_i \rangle^{\mathcal{B}}$ are both random according to the function of SEED protocol (the output is in the secret sharing form), therefore, in the case of a corrupted $\mathcal{A}$, the simulator S_1 just chooses a integers α from $\mathbb{Z}_{2^\ell}$ uniformly at random to simulate $\langle d_i \rangle^{\mathcal{A}}$. In the case of a corrupted $\mathcal{B}$, the simulator S_2 do the same as S_1 to simulate $\langle d_i \rangle^{\mathcal{B}}$. $\mathcal{A}$ and $\mathcal{B}$ call TP to run the function of SSF with input L_1 for outputting the random permutation result $\langle L_2 \rangle^{\mathcal{A}}$ to $\mathcal{A}$ and $\langle L_2 \rangle^{\mathcal{B}}$ to $\mathcal{B}$. The values in $\langle L_2 \rangle^{\mathcal{A}}$ and $\langle L_2 \rangle^{\mathcal{B}}$ are all random according to the function of SSF protocol, therefore, in the case of a corrupted $\mathcal{A}$, the simulator S_1 just chooses the values $\alpha_i (1 \le i \le 2m)$ at random from $\mathbb{Z}_{2^\ell}$ to simulate the values in $\langle L_2 \rangle^{\mathcal{A}}$, and in the case of a corrupted $\mathcal{B}$, the simulator S_2 just do the same as S_1 to simulate the values in $\langle L_2 \rangle^{\mathcal{B}}$. $\mathcal{A}$ and $\mathcal{B}$ call TP to run the function of ADD-CMP with input $\langle \delta_{j,1} \rangle, \langle \delta_{j-1,1} \rangle$ for outputting the result public to $\mathcal{A}$ and $\mathcal{B}$. The case in which the result is assigned to 0 or 1 is randomly given, as $\pi(\pi'(\cdot))$ is randomly selected. Thus, in the case of a corrupted $\mathcal{A}$ or a corrupted $\mathcal{B}$, the simulator just chooses a random integer from $\{0, 1\}$ to simulate this result. According to the composition theory, combining the above analysis and the security of SSF, SEED protocols and ADD-CMP, we can claim that secure query protocol in SSQ-I is secure under the semi-honest adversaries model. □

Proof. (Theorem 6.2) Similar to secure query protocol in SSQ-II, the correctness is easy to see. As for security, we present the security proof in a hybrid model where $\mathcal{A}$ and $\mathcal{B}$ have access to a trusted party (TP) which can realize the function of SSF, SAGSC protocols and ADD-CMP. $\mathcal{A}$ and $\mathcal{B}$ call TP to run the function of SAGSC with input $\langle Q_l \rangle, \langle u_{l,j} \rangle$ for outputting to $\langle \chi_{l,j} \rangle^{\mathcal{A}}$ to $\mathcal{A}$ and $\langle \chi_{l,j} \rangle^{\mathcal{B}}$ to $\mathcal{B}$. The two values are both random according to the function of SAGSC protocol, therefore, in the case of a corrupted $\mathcal{A}$ or $\mathcal{B}$, the simulator just chooses a value α at random from $\mathbb{Z}_{2^\ell}$ to simulate $\langle \chi_{l,j} \rangle^{\mathcal{A}}$ or $\langle \chi_{l,j} \rangle^{\mathcal{B}}$. The remaining security proof is same as that in SSQ-I shown in the proof the Theorem 6.1. □

Next, we demonstrate that SSQ-I can meet the desired privacy requirements, which is trivially guaranteed by the security of our protocols. (1) Data privacy: First of all, we recall that data owners divide the data S_i into two shares by additional secret sharing locally before outsourcing them to the cloud servers. Since the secure query protocol in SSQ-I is secure under semi-honest adversaries model, no information is revealed to $\mathcal{A}$ and $\mathcal{B}$, data privacy can be preserved. (2) Query privacy: The query Q is the secret shared by the user before sending to the cloud servers, and the secure query protocol is secure under the semi-honest model. Therefore, $\mathcal{A}$ and $\mathcal{B}$ obtain no information about Q expect the sequence length, the query privacy is preserved. (3) Hiding data access patterns: In the stage of top-k result return, the key-value pairs $[\langle i \rangle, \langle d_i \rangle]$ are in secret sharing form and permuted by SSF protocol, no one knows $\langle I_j \rangle$' position in the original database. Therefore, $\mathcal{A}$ and $\mathcal{B}$ do not know which data record belongs to the result. Thus, data access patterns are protected from both $\mathcal{A}$ and $\mathcal{B}$. In a similar fashion, we can prove that the desired privacy properties can be achieved in SSQ-II.

C PERFORMANCE ANALYSIS

We first analyze the computational complexity of our five sub-protocols and compare them with existing solutions [26], in which the protocols are same as part of our protocols with a similar security level, and then present the computation complexity of secure query protocol in SSQ-I and SSQ-II. Refer to performance analysis in [26], we only consider the expensive cryptographic primitives including non-XOR gates computation, encryption, decryption, and multiplication in Paillier cryptosystem. The detailed analysis is as following:

SSF Protocol: In Algorithm 1, considering the encryptions performed in line 4 and line 10, the decryptions performed in line 16 and line 20 and the multiplications performed in line 10, SSF_offline totally requires $2n$ encryptions, $2n$ decryptions and n multiplications. In the online phase of SSF protocol, there is no expensive cryptographic primitive.

SBC Protocol: In Algorithm 3, SSF is used twice to permute a sequence with two elements in line 2 and line 3, which incurs 8 encryptions, 8 decryptions, and 4 multiplications. In line 4, an ADD-CMP circuit is used to compare two values. It is important to note that we apply free-XOR [24] technique for garbled circuits optimization. Consequently, one ADD circuit contains ℓ non-XOR gates, one CMP circuit contains ℓ non-XOR gates, so 3ℓ non-XOR gates computation is required in our approach.

SMS Protocol: Based on the permuted sequence generated by calling SSF protocol one time, an ADD-CMP circuit executed $n-1$ times to get the minimum. Therefore, SMS totally requires $2n$ encryptions, $2n$ decryptions n multiplications, and $3\ell(n-1)$ non-XOR gates.

SEED Protocol: As shown in Algorithm 5, SBC is used twice to select the branch and SMS is used once to compute minimum at each iteration and the number of iterations is $n_1 n_2$. Therefore, SEED totally requires $22 n_1 n_2$ encryptions, $22 n_1 n_2$ decryptions $11 n_1 n_2$ multiplications, and $12\ell n_1 n_2$ non-XOR gates computation.

SAGSC Protocol: As we can see from Algorithm 6, the expensive cryptographic primitive in SAGSC protocol is just an execution of EQ-ADD circuit. As one EQ circuit contains $\ell - 1$ non-XOR gates, only $2\ell - 1$ non-XOR gates computation is required in SAGSC protocol.

Secure Query Protocol in SSQ-I and SSQ-II: The computation complexity of secure query protocol in SSQ-I is bounded by $O(m)$ instantiations of SEED, $O(1)$ instantiations of SSF and $O(km)$ instantiations of ADD-CMP. Therefore, the total computation complexity is bounded by $O(mn^2)$ encryptions, decryptions, multiplications, and $O(m\ell(n^2 + k))$ non-XOR gates computation. The computation complexity of Secure Query Protocol in SSQ-II is bounded by $O(mtv)$ instantiations of SAGSC, $O(1)$ instantiations of SSF, and $O(km)$ instantiations of ADD-CMP. Therefore, the total computation complexity is bounded by $O(n)$ encryptions, decryptions, multiplications, and $O(m\ell(tv + k))$ non-XOR gates computation.