Refereed Computation Delegation of Private Sequence Comparison in Cloud Computing

Xu Ma[1], Jin Li[2], and Fangguo Zhang[1]
(Corresponding author: Fangguo Zhang)
School of Information Science and Technology, Sun Yat-Sen University[1]
Guangzhou 510006, P. R. China
School of computer science, Guangzhou University[2]
Guangzhou 510006, P. R. China
(Email: xumasysu@gmail.com, lijin@gzhu.edu.cn, isszhfg@mail.sysu.edu.cn)
(Received June 24, 2013; revised and accepted Sept. 28 & Nov. 26, 2013)

Abstract

Sequence comparison has been widely used in many engineering systems, such as fuzzy keyword search, plagiarism detection, and comparison of gene sequences. However, when the length of the string is extraordinarily long, like the DNA sequence that contains millions of nucleotides, sequence comparison becomes an intractable work, especially when the DNA database is big and the computation resources are limited. Although the generic computation delegation schemes provide a theoretically feasible solution to this problem, it suffers from severe inefficiency when we directly substitute the general function by the sequence comparison function. In this paper, we focus on refereed computation delegation of sequence comparison and present the refereed computation delegation scheme of sequence comparison using multiple servers. In our scheme, the user can detect the dishonest servers and get the correct answer as long as there is one honest server. The direct application of our scheme is DNA sequence comparison of big gene database in medical system. Meanwhile, our scheme satisfies the security requirement of sequence privacy against the malicious adversaries. Moreover, since neither the fully homomorphic encryption nor the complicated proof systems are used for the problem generation and result verification, our solution clearly outperforms the existing schemes in terms of efficiency. The computation complexity of the user is reduced from $O(mn)$ to $O(\log^2(mn))$, where m,n are the length of the sequences.

Keywords: Privacy, refereed computation delegation, sequence comparison

1 Introduction

Sequence comparison can be viewed as the string editing problem, i.e., computing the distance between two strings. The edit distance is one of the most widely used notions of similarity: it is the least-cost operation set of deletions, insertions and substitutions required to transform one string into another. Sequence comparison is widely used in many engineering systems, such as fuzzy keyword search [11, 28], plagiarism detection, and comparison of gene sequences [9]. However, the computation complexity of sequence comparison is $O(mn)$, where m and n are the respective length of the strings. When the length of the string is extraordinarily long, like the DNA sequence that contains millions of nucleotides, sequence comparison becomes an intractable work, especially for resource limited devices.

Generally, such computation expensive tasks can deploy the so-called "computation delegation" to accomplish the tasks efficiently. Computation delegation [1, 15, 16] considers a scenario where one party, the delegator who is computationally weak, wishes to delegate the computation of a function f on various inputs $x_1, x_2, \cdots, x_k$ to one or more servers who are computationally strong. However, the servers are not fully trusted, the basic security requirements of computation delegation are verifiability and efficiency, which require that the delegator should be able to verify the correctness of the values returned by the worker. Moreover, the verification process should require substantially less computation efforts than computing $f(x)$ from scratch. In addition, an important property of verifiable computation is privacy, which enables the delegator to hide some private information from the worker. As we know, individual DNA and protein sequences are highly sensitive and vulnerable to re-identification even when anonymized. Therefore, the outsourcing technique should enable the desired computation without revealing any information about the sequences to the parties carrying out the computation.

However, if we trivially apply the traditional computation delegation scheme to sequence comparison, the resulting scheme becomes severely inefficient. The reason

is that most of the previous works are based on complex cryptographic tools, such as fully homomorphic encryption [17] and proof systems [6, 19].

In [4], Atallah et al. proposed a secure outsourcing scheme of sequence comparison. However, the protocol was only proved to be privacy secure in semi-honest adversary model, and the most significant security requirement of verifiability was not satisfied. Apart from [4], most of the previous works related to sequence comparison have been done in the framework of two-party computation model [10], in which two parties with private inputs wish to jointly compute some function of their inputs while preserving certain properties like privacy and correctness. These works are quite related to private pattern matching, which is out of the scope of this paper.

Our contributions. In this paper, we present a new computation delegation model, which is called refereed computation delegation of sequence comparison, using multiple servers. Our contributions are two-fold:

- The user can detect the dishonest servers and get the correct answer as long as there is at least one honest server.
- Our scheme satisfies the security requirement of input/output privacy against the malicious servers.

In multi-server model, one trivial method to realize the verifiability is: the user picks N different servers and asks each of those to execute his programme and return the output. Now, the user takes the plurality value of those answers to be the correct answer. As long as there is a majority of honest servers, the user gets the correct answer. The main drawback of this approach is the need for an honest majority of servers.

To get better performance and abate the assumption of a plurality of honest servers, we propose a new approach which is called refereed computation delegation of sequence comparison based on Canetti's computation delegation scheme [12]. In our scheme, the user runs like a referee who supervises the servers. The servers do the computation and return the result together with a commitment of the result back to the user. In the verification process, when the user detects inconsistency between the returned results, the process of consistency proof and verification are activated. After that, the user can get the correct result by performing only a *single* step of the computation of sequence comparison. Specially, our scheme is implementable suppose that there are only two servers, one of which is honest.

As for efficiency, since neither the fully homomorphic encryption nor the complicated proof systems are used for the problem generation and result verification, our solution clearly outperforms the existing schemes in terms of efficiency. In detail, the computation load of a server is $O(cmn)$, and the computation complexity of the user is reduced from $O(mn)$ to $O(\log^2(mn))$, where m,n are the length of the sequences and c is a constant.

Related work. Atallah et al. proposed a secure outsourcing scheme of sequence comparison in [4] using two non-collusion servers. However, the security model of the protocol is semi-honest and the security requirement of verifiability was not mentioned. In [14,24,38], the authors studied the sequence comparison in the framework of two-party computation model, in which two parties with private inputs wish to jointly compute some function of their inputs while preserving the security requirements like privacy and correctness. These works are quite related to private pattern matching [14, 21, 22, 39], where party P_1 holds a pattern and party P_2 holds a text. The goal of P_1 is to learn where the pattern appears in the text, without revealing it to P_2 and learning anything else about P_2's text.

Computation delegation has received widespread attention due to the rise of cloud computing [13,37], where businesses buy computing power from a service, rather than purchasing and maintaining their own computing resources. Another motivation of computation delegation is the proliferation of mobile devices, such as netbooks and smart phones. Due to the computation and storage limitations, sometimes it is desirable to off-load heavy computations, such as cryptographic operations, or photo manipulation, to the cloud server. However, the cloud server is not fully trusted and sometimes the applications outsourced to the cloud are so critical that it is imperative to keep the original data private and rule out accidental errors during the computation.

Previous research on computation delegation can be classified into two categories: 1) the generic computation delegation [1, 15, 16]: it can be applied for arbitrary functions; 2) the concrete computation delegation [2, 5, 7, 23, 25, 29, 30]: they are designed for some specific functions, such as polynomial evaluation and linear algebra. In the generic model, most of the previous works are based on fully homomorphic encryption [17] and proof systems, such as interactive proofs [6, 19], efficient arguments based on probabilistically checkable proofs (PCP) [26, 27], CS proofs [32]and the muggles proofs in [18]. The complex cryptographic tools used in the generic model result in inefficiency when applying these protocols to some concrete functions.

For the computation delegation of specific functions, plenty of research works have been proposed. Benjamin and Atallah [8] addressed the problem of secure outsourcing for widely applicable linear algebra computations. However, the proposed protocols required the expensive operations of homomorphic encryptions. Atallah and Frikken [1] further studied this problem and gave improved protocols based on the so-called weak secret hiding assumption. Benabbas et al. [7] presented the first practical computation delegation scheme for high degree polynomial functions based on the approach of [16]. In 2011, Green et al. [20] proposed new methods for efficiently and securely outsourcing decryption of attribute-based encryption (ABE) ciphertexts. Based on this work, Parno et al. [35] showed a construction of a multi-function computation delegation scheme.

Organization. The paper is organized as follows. In

Section 2, we give a brief description of the preliminaries which will be used in the following sections, In Section 3, we present the system model and security definition of our scheme. The construction of our scheme is presented in Section 4. In Section 5, we give the efficiency analysis and security proof of our scheme. Finally, we conclude in Section 6.

2 Preliminaries and Tools

2.1 Edit Distance

We now precisely give the definition of edit distance [40]. Consider a finite alphabet set Σ whose elements will be used to construct strings. Let $\mathbb{C}_I$, $\mathbb{C}_D$, $\mathbb{C}_S$ be finite sets whose elements are finite integers. And let the function $I : \Sigma \to \mathbb{C}_I$ be insertion cost function, i.e., $I(a)$ is the cost of inserting an element $a \in \Sigma$ to a given string. Similarly, define the deletion cost function as $D : \Sigma \to \mathbb{C}_D$, i.e., $D(a)$ is the cost of deleting an element $a \in \Sigma$ from a given string. And define the substitution cost function as $S : \Sigma \times \Sigma \to \mathbb{C}_S$, i.e., $S(a, b)$ is the cost of replacing an element $a \in \Sigma$ in a given string by an element $b \in \Sigma$.

If we let λ be a string of length n, $\lambda = \lambda_1, \lambda_2, \cdots, \lambda_n$, and μ be a string of length m, $\mu = \mu_1, \mu_2, \cdots, \mu_m$, both are strings over alphabet set Σ. As mentioned above, there are three allowed edit operations to be operated on λ, insertion of an element, deletion of an element and substitution of one element by another. Each sequence of operations that transforms λ into μ has an aggregate cost associated with it, which is equal to the sum of the costs of the operations in it. The least-cost of such sequences is the edit distance.

We now give a brief review of the standard dynamic programming for computing edit distance [40]. Let $M(i, j)$ $(0 \le i \le n, 0 \le j \le m)$be the minimum cost of transforming the prefix of λ of length i into the prefix of μ of length j, i.e., of transforming $\lambda_1, \lambda_2, \cdots, \lambda_i$ into $\mu_1, \mu_2, \cdots, \mu_j$. Then $M(0, 0) = 0$, $M(0, j) = \sum_{k=0}^{j} I(\mu_k)$ for $1 \le j \le m$, and $M(i, 0) = \sum_{k=0}^{i} D(\lambda_k)$, for $0 \le i \le n$. For positive i and j, we have

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

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

2.2 Merkle Hash Tree

Merkel Hash Tree (MHT) [31] is a common primitive that allows one to hash a long string of n characters in a way that the hash can later be used to reveal any part of the string and supply a short proof of consistency. The construction of MHT is based on the collision-resistant hash function, and given a collision-resistant hash function H and string str of length n, the tree has n leaf nodes where leaf node i has the value of $H(str[i])$, $str[i]$ is the i-th character of str. The next level has the values of $H(H(str[i])||H(str[i+1]))$, for $i = 1, 3, \cdots, n-1$, and so on for the other levels. The proof of consistency for character i consists of $H(str[i])$ and all the sibling hash values of the nodes along the path from the root to the leaf node $H(str[i])$.

Given a MHT of a string str, denote by $MH_{root}(str)$ the value of the root, by $MH_{proof}(str, i)$ the proof of consistency for the i-th character, and by $VerMHP(root, i, str_i, p)$ the verification function that given a claimed proof $p = MH_{proof}(str, i))$ outputs true if p is valid and false otherwise. Note that the size of the proof is $\log n$ and the complexity of verification function is $O(\log n)$.

3 System Model and Security Definitions

3.1 System Model

A refereed computation delegation $\mathcal{RCD}$ [12] for a function f is a protocol between a user (or referee) R and N servers $S_1, S_2, \cdots, S_N$. The difference between traditional computation delegation model and $\mathcal{RCD}$ is that the user acts like a referee in the delegation procedure. The user is able to detect the dishonest server when there is a dispute. The user and the servers receive the input x. The servers compute the function in parallel and claim different result of the computation of $f(x)$, and the user should be able to detect the dishonest servers and determine the correct $f(x)$ with overwhelming probability as long as there is at least one honest server. Formally, for any input x and for all $i \in \{1, \cdots, N\}$, if S_i is honest, then for any potentially dishonest $S_1^*, \cdots, S_{i-1}^*, S_{i+1}^*, \cdots, S_N^*$, the output of R is $f(x)$ with probability at least $1 - \varepsilon$, where ε is negligible. An optional security requirement of $\mathcal{RCD}$ is I/O privacy. In the following subsection, we will give a formal definitions of the security requirements.

Firstly, we retrospect the traditional verifiable computation model. In detail, the traditional verifiable computation scheme consists of four algorithms defined below (KeyGen,ProbGen, Compute, Verify):

- $(pk, sk) \leftarrow$ KeyGen(f,λ): *Based on the security parameter λ, the randomized key generation algorithm generates a public key pk that encodes the target function $f(\cdot)$, which is used by the cloud server to compute $f(\cdot)$. It also computes a matching secret key sk, which is kept secret by the user U.*

- $(\sigma_x, \tau_x) \leftarrow \mathsf{ProbGen}_{sk}(x)$: *The problem generation algorithm uses the secret key sk to encode the input x as a public value σ_x, which is given to the cloud server*

to compute with, and a secret value τ_x, which is kept private by the user U.

- $\sigma_y \leftarrow \mathsf{Compute}_{pk}(\sigma_x)$: *Using the user's public key pk and the encoded input σ_x, the server computes and outputs an encoded version of the result $y = f(x)$.*

- $y \vee \perp \leftarrow \mathsf{Verify}_{sk}(\sigma_y, \tau_x)$: *Using the secret key sk and the secret "decoding" value τ_x, the verification algorithm converts the cloud server's encoded output into the output of $y = f(x)$ or $\perp$ indicating that σ_y does not represent the valid output of $f(\cdot)$ on x.*

Now we give a formal description of refereed computation delegation. In detail, a refereed computation delegation scheme $\mathcal{RCD}$=(KeyGen, ProbGen, Compute, Verify) consists of four algorithms as defined below, which is the same as that in the verifiable computation [7,16], except that now N servers receive the same input from the user and return the results to the user.

- $(pk, sk) \leftarrow \mathsf{KeyGen}(\mathrm{f}, \kappa)$: *Based on the security parameter κ, the randomized key generation algorithm generates a public key/secret key pair pk/sk for the function $f(\cdot)$. The public key is provided to the servers, while the secret key is kept private by the user.*

- $(\sigma_x, \tau_x) \leftarrow \mathsf{ProbGen}_{sk}(x)$: *The problem generation algorithm uses the secret key sk to encode the input x as a public value σ_x, which is given to all the cloud servers to compute with, and a secret value τ_x, which is kept private by the user U.*

- $\sigma_y \leftarrow \mathsf{Compute}_{PK}(\sigma_x)$: *Using the user's public key pk and the encoded input σ_x, the servers compute and output an encoded version of the result $y = f(x)$, respectively.*

- $y \cup \perp \leftarrow \mathsf{Verify}_{sk}(\sigma_y)$: *Using the secret key sk and the secret "decoding" value τ_x, the verification algorithm converts the cloud servers' encoded output $y*$. Note that $y* \neq y$ if the server is dishonest. Then U verifies the correctness of $y*$ and obtains the correct result as long as there is at least one honest server.*

The basic efficiency requirement of a $\mathcal{RCD}$ scheme is that the time to encode the input and verify the output must be smaller than the time to compute the function from scratch, and the complexity of the servers is polynomial in the complexity of evaluating f. Formally, A $\mathcal{RCD}$ can be outsourced if it permits efficient problem generation and verification. This implies that for any input x and output σ_y, the time required for $\mathsf{ProbGen}_{sk}(x)$ plus the time required for $\mathsf{Verify}_{sk}(\tau_x, \sigma_y)$ is smaller than T, where T is the time to compute the function $f(x)$ from scratch.

3.2 Security Requirements

A refereed computation delegation scheme should be both correct and secure. Intuitively, a $\mathcal{RCD}$ scheme is correct if the user always outputs the correct result $f(x)$ as long as there is at least one honest cloud server. In the following experiments, A denotes the set of malicious cloud servers who are allowed to collude with each other, of which the size is at most $N-1$. Note that in the refereed delegation of computation, all the servers receive the same input, therefore, the oracle answer for an adversary set A just contains one answer. And the members of the adversary set A will output the same result in order to improve the probability of successful attack. Thus, the adversary set A can be viewed as one party.

Experiment $\mathbf{Exp}_A^C[\mathcal{RCD}, f, \kappa]$
$(pk, sk) \leftarrow \mathsf{KeyGen}(f, \kappa)$;
For $i = 1, \cdots, l = poly(\kappa)$
$x_i \leftarrow A(x_1, \sigma_{x_1}, \beta_1, \cdots, x_{i-1}, \sigma_{x_{i-1}}, \beta_{i-1})$;
$(\sigma_{x_i}, \tau_{x_i}) \leftarrow \mathsf{ProbGen}_{sk}(x_i)$;
$\sigma_{y_i} \leftarrow A(pk, x_1, \sigma_{x_1}, \beta_1, \cdots, x_{i-1}, \sigma_{x_{i-1}}, \beta_{x_{i-1}}, \sigma_{x_i})$;
$\beta_i = \mathsf{Verify}_{sk}(\tau_{x_i}, \sigma_{y_i})$;
$x \leftarrow A(pk, x_1, \sigma_{x_1}, \beta_1, \cdots, x_l, \sigma_{x_l}, \beta_l)$
$(\sigma_x, \tau_x) \leftarrow \mathsf{ProbGen}_{sk}(x)$;
$\sigma_y \leftarrow A(pk, x_1, \sigma_{x_1}, \beta_1, \cdots, x_l, \sigma_{x_l}, \beta_l, \sigma_x)$;
$\hat{y} \leftarrow \mathsf{Verify}_{sk}(\tau_x, \sigma_y)$;
$\hat{y} \neq f(x)$, output 1, else 0;

In the above experiment, the malicious servers are given oracle access to generate the encoding of multiple problem instances, and also oracle access to the result of the verification algorithm on arbitrary strings on those instances. The adversary succeeds if they convince the user to output wrong result for a given input value. Our goal is to make the adversary succeed only with negligible probability.

Correctness. For a refereed computation delegation scheme $\mathcal{RCD}$, we define the advantage of a set of adversaries A in the experiment above as:

$$Adv_A(\mathcal{RCD}, f, \kappa) = Pr[\mathbf{Exp}_A^C[\mathcal{RCD}, f, \kappa] = 1]$$

A refereed computation delegation scheme $\mathcal{RCD}$ is correct if for any function $f \in \mathcal{F}$, and for any adversary set A whose members run in probabilistic polynomial time,

$$Adv_A(\mathcal{RCD}, f, \kappa) \leq neg(\kappa)$$

where $neg(\cdot)$ is a negligible function of its input.

While the basic security requirements of correctness protects the integrity of $\mathcal{RCD}$, it is also desirable to protect the input and output of the refereed computation delegation scheme against the malicious servers. Below, we define the input/output privacy based on a typical indistinguisability argument that guarantees that no information about the inputs/outputs is leaked.

Intuitively, a refereed verifiable computation scheme is input private when the public outputs of the problem generation algorithm ProbGen over two different inputs are indistinguishable. In the following experiment, the

adversaries are allowed to request the encoding of any input they desires. The $\mathsf{PubProbGen}$ returns the public parameter σ_x to the adversary.

Experiment $\mathbf{Exp}_A^{IPriv}[\mathcal{RCD}, f, \kappa]$
$(pk, sk) \leftarrow \mathsf{KeyGen}(f, \kappa)$
$(x_0, x_1) \leftarrow A^{\mathsf{PubProbGen}(\cdot)}(pk)$
$(\sigma_0, \tau_0) \leftarrow \mathsf{ProbGen}_{sk}(x_0)$
$(\sigma_1, \tau_1) \leftarrow \mathsf{ProbGen}_{sk}(x_1)$
$b \leftarrow_R \{0, 1\}$
$\hat{b} \leftarrow A^{\mathsf{PubProbGen}_{sk}(\cdot)}(pk, x_0, x_1, \sigma_b)$
If $\hat{b} = b$, output1, else 0.

Input Privacy. For a refereed delegation of computation scheme, we define the advantage of an adversary set A in the experiment above as :

$$Adv_A^{IPriv}(\mathcal{RCD}, f, \kappa) = |Pr[\mathbf{Exp}_A^{IPriv}[\mathcal{RCD}, f, \kappa] = 1] - \frac{1}{2}|.$$

A refereed computation delegation scheme is input private for a function f, if for any adversary set A whose members run in probabilistic polynomial time,

$$Adv_A^{IPriv}(\mathcal{RCD}, f, \kappa) \leq neg(\kappa)$$

where $neg()$ is a negligible function of its input.

A similar definition can be made for output privacy. We consider the following the experiment. During the process, the adversaries are allowed to request the output of the algorithm $y \cup \perp \leftarrow \mathsf{Verify}$. In the challenged phase, the adversary has to submit two encoded outputs σ_0 and σ_1. After the adversaries have chosen the challenged output, they can continue the query process. The experiment ideally simulates the execution process of the refereed computation delegation scheme. The oracle $\mathsf{OVerify}$ returns the output of the algorithm $y \cup \perp \leftarrow \mathsf{Verify}$ to the adversaries.

Experiment $\mathbf{Exp}_A^{OPriv}[\mathcal{RCD}, f, \lambda]$
$(pk, sk) \leftarrow \mathsf{KeyGen}(f, \lambda)$
$(\sigma_{y_0}, \sigma_{y_1}) \leftarrow A^{\mathsf{OVerify}}_{\mathsf{PubProbGen}}(pk, f, \lambda)$
$y_0 \leftarrow \mathsf{Verify}(\sigma_{y_0}, f, \lambda)$
$y_1 \leftarrow \mathsf{Verify}(\sigma_{y_1}, f, \lambda)$
$b \leftarrow \{0, 1\}$
$\hat{b} \leftarrow A^{\mathsf{OVerify}}_{\mathsf{PubProbGen}}(f, pk, y_b)$
if $\hat{b} = b$, output 1, else 0.

Output Privacy. For a refereed computation delegation scheme $\mathcal{RCD}$, we define the advantage of an adversary set A in the experiment above as:

$$Adv_A^{Opriv}(\mathcal{RCD}, f, \kappa) = |Pr[\mathbf{Exp}_A^{Opriv}[\mathcal{RCD}, f, \kappa] = 1 - \frac{1}{2}|$$

A refereed delegation of computation scheme is output private if for the legal encoded output set σ, and for any adversary set A whose members run in probabilistic polynomial time,

$$Adv_A^{Opriv}(\mathcal{RCD}, f, \kappa) \leq neg(\kappa)$$

where $neg()$ is a negligible function of its input.

4 Construction

Main Idea. In the following sections, we only discuss the case where there are two pairs of servers (W_{11}, W_{12}) and (W_{21}, W_{22}), of which one pair of servers is honest. In the following subsections, we will explain how to extend it to N server model. Suppose the user has two sequences λ and μ over some finite alphabet $\Sigma = \{0, \cdots, \sigma - 1\}$ and he wants to delegate the sequence comparison of λ and μ to the cloud servers. We assume each pair of cloud servers cooperatively execute the protocol, but they do not collude with each other. The reason why we use a pair of cloud servers as a server unit will be explained in the following sections.

As stated in Figure 1., the user starts by splitting the sequence λ into λ' and λ'' such that λ' and λ'' are over the same alphabet $\Sigma = \{0, 1, \cdots, \sigma - 1\}$ and $\lambda_i = \lambda_i' + \lambda_i''$ for all $1 \leq i \leq n$. To split λ, the user can first generate a random sequence λ' over the alphabet Σ of length n, and then set $\lambda_i'' = \lambda_i - \lambda_i' \bmod \sigma$, for all $1 \leq i \leq n$. Similarly, the user splits μ into μ' and μ'' such that $\mu_j = \mu_j' + \mu_j''$, for all $1 \leq j \leq m$. Then, λ', μ' are sent to W_{11} and λ'', μ'' are sent to W_{12}. The servers W_{11}, W_{12} collaboratively compute the the edit distance $M(n, m)$ of λ and μ in an additively split fashion. That is, W_{11} and W_{12} each maintain a matrix $M^{(1)'}$ and $M^{(1)''}$ such that $M(i, j) = M^{(1)'}(i, j) + M^{(1)''}(i, j)$ for $1 \leq i \leq n$, $1 \leq j \leq m$. In addition, W_{11} and W_{12} each construct the Merkle Hash Tree for all the values of the matrixes $M^{(1)'}$ and $M^{(1)''}$, and return the result $M^{(1)'}(n, m)$, $M^{(1)''}(n, m)$ together with the value $MH_{root}(M^{(1')}(n, m))$, $MH_{root}(M^{(1'')}(n, m))$, which is the root value of the Merkle Hash Tree for their respective result. The second pair of servers (W_{21}, W_{22}) do the same as described above. Thereafter, the user runs the consistency proof with $(W_{11}, W_{12}), (W_{21}, W_{22})$, and outputs the correct result if at least one pair of servers are honest.

As described in Section 2, a refereed computation delegation of sequence comparison scheme consists of four algorithms (KeyGen, $\mathsf{ProbGen}$, $\mathsf{Compute}$, Verify), which will be formally defined below. The framework of our scheme is presented in Figure. 1.

- $(pk, sk) \leftarrow \mathsf{KeyGen}(\mathrm{f}, \kappa)$: Based on the security parameter κ, the randomized key generation algorithm generates the additively homomorphic encryption [**?**, 33, 34]key pair (pk, sk) for every cloud server and randomly selects a hash function H.
- $(\sigma_x, \tau_x) \leftarrow \mathsf{ProbGen}(x)$: On private input $x = (\lambda, \mu)$, the user splits the sequence λ into λ' and λ'', such that $\lambda = \lambda' + \lambda'' \bmod \sigma$, and similarly splits

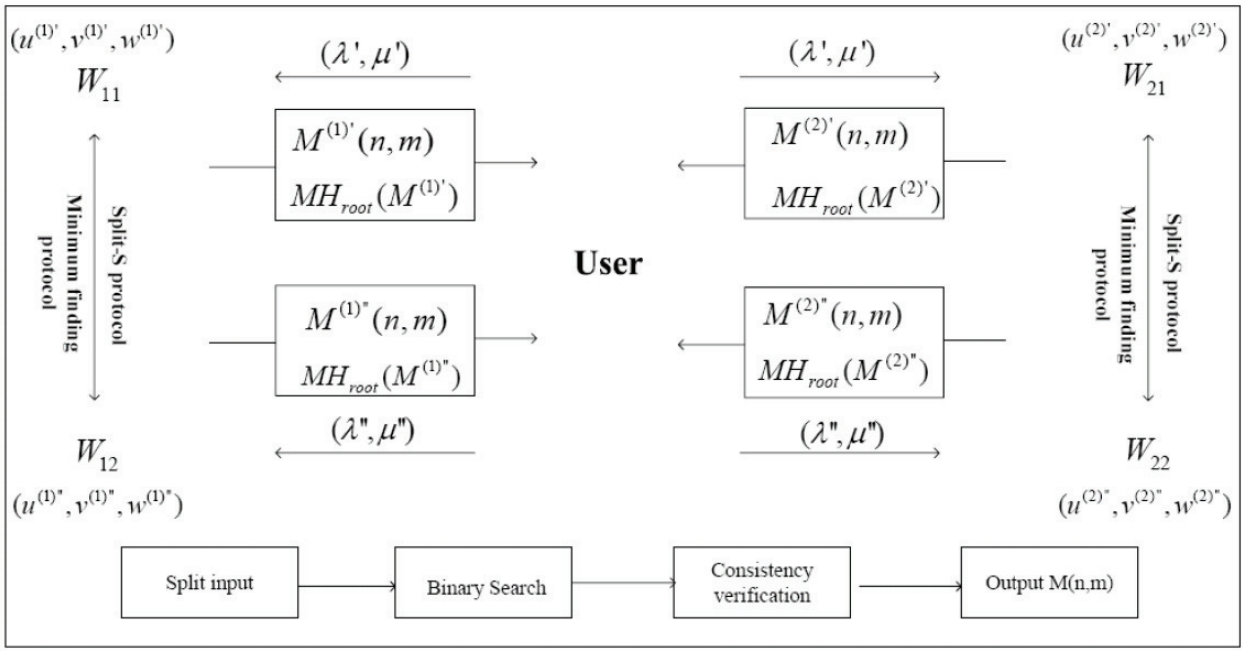

Figure 1: Framework of refereed computation delegation of private sequence comparison

μ into μ' and μ'' such that $\mu = \mu' + \mu''$ mod σ. Meanwhile, U generates two vectors of random numbers, $a = (a_1, \cdots, a_n)$ and $b = (b_1, \cdots, b_m)$. Then U computes two vectors $c = (c_1, \cdots, c_n)$ and $d = (d_1, \cdots, d_m)$ where $c_i = \sum_{k=1}^{i} D(\lambda_k) - a_i$, for $1 \le i \le n$, and $d_j = \sum_{k=1}^{j} I(\mu_k) - b_j$, for $1 \le j \le m$. The outputs of the algorithm is set to be $(\sigma_x, \tau_x)=((\lambda', \lambda'', \mu', \mu'', a, b, c, d), (\lambda, \mu))$. Then U sends λ', μ', b, c to W_{k1}, and sends λ'', μ'', a, d, to W_{k2}, $k = 1, 2$.

- $\sigma_y \leftarrow \mathsf{Compute}(\sigma_x)$: As described in Figure 2. (W_{k1}, W_{k2}), $k = 1, 2$, collaboratively compute $M(i,j)$ for all $1 \le i \le n$, $1 \le j \le m$. Note that W_{k1} owns $M^{(k)}(i,j)'$ and W_{k2} owns $M^{(k)}(i,j)''$, such that $M(i,j) = M^{(k)}(i,j)' + M^{(k)}(i,j)''$.

  The protocol specifications are described as follows:

  1) W_{k1} and W_{k2} first initialize their respective matrix, and then cooperatively compute $(u^{(k)'}, v^{(k)'}, w^{(k)'})$ and $(u^{(k)''}, v^{(k)''}, w^{(k)''})$.
  2) After the minimum finding protocol [3], W_{k1} gets $M^{(k)'}(i,j)$ and W_{k2} gets $M^{(k)''}(i,j)$.
  3) Then W_{k1} and W_{k2} each construct the Merkel Hash Tree for the result matrices $M^{(k)'}$ and $M^{(k)''}$ respectively and return the result and root value to the user. Thus, the output σ_y=$(M^{(k)'}(n,m), MH_{root}(M^{(k)'}), M^{(k)''}(n,m), MH_{root}(M^{(k)''}))$, $k = 1, 2$.

  Note that γ' and γ'' are computed through the following split-S protocol.

  1) W_{k1} generates a $\sigma \times \sigma$ table $\hat{S}$ with $\hat{S}(r,l)$ equals to $E_{pk_{k1}}(S(r + \lambda'_i mod\ \sigma, l))$ and sends it to W_{k2}.
  2) For $1 \le j \le m$, W_{k2} extracts the λ''_i-th row of $\hat{S}$ as a vector v, $v_l = E_{pk_{k1}}(S(\lambda''_i + \lambda'_i mod\ \sigma, l)) = E_{pk_{k1}}(S(\lambda_i, l))$. Then W_{k2} circularly left-shift the vector v μ''_j positions and updates v with a random number γ'', as a result $v_l = E_{pk_{k1}}(S(\lambda_i, \mu''_j + l) * E_{pk_{k1}}(-\gamma'')) = E_{pk_{k1}}(S(\lambda_i, \mu''_j + l) - \gamma'')$.
  3) W_{k1} uses 1-out-of-m oblivious transfer protocol [36]to get the μ'_j-th item of v and decrypts it as $\gamma' = S(\lambda_i, \mu''_j + \mu'_j) - \gamma'' = S(\lambda_i, \mu_j) - \gamma''$.

- $y \cup \perp \leftarrow \mathsf{Verify}_{sk}(\sigma_y)$: U first verifies whether $M^{(1)}(n,m)' + M^{(1)}(n,m)'' = M^{(2)}(n,m)' + M^{(2)}(n,m)''$ or not. In case the equation holds, the answer is correct. Else, the user continues to a binary search as described as follows.

Remark. Assume that the split edit distance matrix is reordered in row-major principle as a vector $\hat{m}$ when the cloud servers construct the Merkle Hash Tree for them. We use variable n_g to denote the good positions which means that $\hat{m}^{(1)'}(n_g)+\hat{m}^{(1)''}(n_g)=\hat{m}^{(2)'}(n_g)+\hat{m}^{(2)''}(n_g)$, and use variable n_b to denote the bad positions. When the binary search ends, U ask W_{11} and W_{12} for the consistency proof at the positions n_g and n_b. If the verification process returns true, U outputs $M(n,m) = M^{(1)}(n,m)' + M^{(1)}(n,m)''$. Otherwise, he outputs $M(n,m) = M^{(2)}(n,m)'+M^{(2)}(n,m)''$. The specifications are presented as follows:

1) U initializes position variables as $n_g = 1$, $n_b = mn$ and asks W_{k1} and W_{k2}, $k = 1, 2$, for the mid-th value

For $k = 1, 2$

Initialization:

W_{k1} sets $M^{(k)'}(0, j) = b_j$, for $1 \le j \le m$, and sets $M^{(k)'}(i, 0) = c_i$, for $1 \le i \le n$.

W_{k2} sets $M^{(k)''}(0, j) = d_j$, for $1 \le j \le m$, and sets $M^{(k)''}(i, 0) = a_i$, or $1 \le i \le n$.

For $1 \le i \le n$, $1 \le j \le m$

(1). W_{k1} computes $u^{(k)'} = M^{(k)'}(i-1, j) + M^{(k)'}(i, 0) - M^{(k)'}(i-1, 0)$
$= M^{(k)'}(i-1, j) + D(\lambda_i) - a_i + a_{i-1}$.

W_{k2} computes $u^{(k)''} = M^{(k)''}(i-1, j) + M^{(k)''}(i, 0) - M^{(k)''}(i-1, 0)$
$= M^{(k)''}(i-1, j) + a_i - a_{i-1}$.

$u^{(k)'} + u^{(k)''} = M^{(k)'}(i-1, j) + M^{(k)''}(i-1, j) = M(i-1, j) + D(\lambda_i)$.

(2). W_{k1} computes $v^{(k)'} = M^{(k)'}(i, j-1) + M^{(k)'}(0, j) - M^{(k)'}(0, j-1)$
$= M^{(k)'}(i, j-1) + b_j - b_{j-1}$.

W_{k2} computes $v^{(k)''} = M^{(k)''}(i, j-1) + M^{(k)''}(0, j) - M^{(k)''}(0, j-1)$
$= M^{(k)''}(i, j-1) - b_j + b_{j-1} + I(\mu_j)$.

$v^{(k)'} + v^{(k)''} = M^{(k)'}(i, j-1) + M^{(k)''}(i, j-1) = M(i, j-1) + I(\mu_j)$.

(3). W_{k1} sets $w^{(k)'} = M^{(k)'}(i-1, j-1) + \gamma'$.

W_{k2} sets $w^{(k)''} = M^{(k)''}(i-1, j-1) + \gamma''$.

$w^{(k)'} + w^{(k)''} = M(i-1, j-1) + S(\lambda_i, \mu_j)$

(4). After the implementation of minimum finding protocol, W_{k1} and W_{k2} get $M^{(k)'}(i, j)$, $M^{(k)''(i,j)}$ respectively, such that

$$M^{(k)'}(i, j) + M^{(k)''}(i, j) = min \begin{pmatrix} M(i-1, j-1) + S(\lambda_i, \mu_j) \\ M(i-1, j) + D(\lambda_i) \\ M(i, j-1) + I(\mu_i) \end{pmatrix}$$

Figure 2: Specifications for the algorithm Compute

of the result vector and the consistency proof of the values, where $mid = (n_g + n_b)/2$. If $\hat{m}^{(1)'}(mid) + \hat{m}^{(1)''}(mid) = \hat{m}^{(2)'}(mid) + \hat{m}^{(2)''}(mid)$, he sets $n_g = mid$, otherwise, he sets $n_b = mid$. U continues the binary search in that way till he gets $n_b = n_g + 1$.

2) W_{11} returns the consistency proof

$$p_{1g} = MH_{poof}(\hat{m}^{(1)'}, n_g), p_{1b} = MH_{poof}(\hat{m}^{(1)'}, n_b),$$

W_{12} returns

$$p_{2g} = MH_{poof}(\hat{m}^{(1)''}, n_g), p_{2b} = MH_{poof}(\hat{m}^{(1)''}, n_b).$$

3) U runs

$$VerMHP(MH_{root}(\hat{m}^{(1)'}), n_g, \hat{m}^{(1)'}(n_g), p_{1g})$$
$$VerMHP(MH_{root}(\hat{m}^{(1)'}), n_b, \hat{m}^{(1)'}(n_b), p_{1b})$$
$$VerMHP(MH_{root}(\hat{m}^{(1)''}), n_g, \hat{m}^{(1)''}(n_g), p_{2g})$$
$$VerMHP(MH_{root}(\hat{m}^{(1)''}), n_b, \hat{m}^{(1)''}(n_b), p_{2b})$$

to verify the consistency proof. If either proof is invalid, the cloud servers W_{k1} and W_{k2} are marked as dishonest. Otherwise, U proceeds as follows.

4) Suppose that $\hat{m}^{(1)'}(n_g)$ is equivalent to $M^{(1)'}(\alpha, \beta)$, then $\hat{m}^{(1)'}(n_g - m)$ and $\hat{m}^{(1)'}(n_g - m + 1)$ are equivalent to $M^{(1)'}(\alpha - 1, \beta)$, $M^{(1)'}(\alpha - 1, \beta + 1)$. As described above, U asks W_{11} and W_{12} for consistency proof at the position $n_g - m$ and $n_g - m + 1$, and then computes

$$M(\alpha - 1, \beta) = M^{(1)'}(\alpha - 1, \beta) + M^{(1)''}(\alpha - 1, \beta)$$

$$M(\alpha-1, \beta+1) = M^{(1)'}(\alpha-1, \beta+1)+M^{(1)''}(\alpha-1, \beta+1).$$

Now that U owns $M(\alpha, \beta)$, $M(\alpha - 1, \beta)$, and $M(\alpha - 1, \beta + 1)$, he can compute $M(\alpha, \beta + 1)$ by himself

using the dynamic programming algorithm. Then U verifies whether $M(\alpha,\beta+1) = M^{(1)'}(\alpha,\beta+1) + M^{(1)''}(\alpha,\beta+1)$ holds (Remember that $M^{(1)'}(\alpha,\beta+1) = \hat{m}^{(1)'}(n_b)$, $M^{(1)''}(\alpha,\beta+1) = \hat{m}^{(1)''}(n_b)$. If the equation holds, U outputs the value of $M(n,m) = M^{(1)'}(n,m) + M^{(1)''}(n,m)$. Otherwise, $M(n,m) = M^{(2)'}(n,m) + M^{(2)''}(n,m)$.

Extensions. Here we show a trivial method of how, given refereed computation delegation scheme with two servers, one can construct a refereed delegation of computation with N servers, where we only need to assume that at least one of them if honest. The idea is to execute the refereed delegation of computation with two servers between each pair of servers. By the soundness of the refereed delegation of computation, with high probability there exists an honest server pair W_i that convinces the user in all of his games. The user outputs the claimed result of W_i.

In addition to the trivial method for extending the protocol to N servers, we can extend this specific protocol also in the following way. Our protocol can be executed with all servers, where the user marks the intermediate value as a good value only if all answers for this position match. At the end of the binary search, the user checks if the computation is consecutive for each one of the servers. After the execution of this protocol, at least one pair of malicious servers will be caught lying and will be declared as a cheater. The user continues to the next round with the other servers, again, executes the protocol to find at least one cheater and then excludes him (or them) from the next rounds. The protocol ends when all the remaining servers agree on the output.

5 Efficiency and Security Analysis

5.1 Efficiency Analysis

First of all, our protocol, which works in the multi-server model, is qualitatively more practical than known techniques for computation delegation in single-server setting. As we know, all known protocols rely either on arithmetization and PCP techniques [18], or rely on fully homomorphic encryption [15,16]. Neither approach is currently viable in practice as a result of low efficiency.

Our protocol neither utilizes the complex proof systems nor fully homomorphic encryption, it is very efficient on both the user side and server side. The computation and communication complexity of splitting the sequence λ and μ and sending the result shares is $O(m+n)$, which is also the lower bound of problem generation algorithm. In the verification procedure, the user searches for inconsistencies between the intermediate values of the two servers' computations. Note that the binary search algorithm ends within $O(\log\ mn)$ steps, where $\log\ m$ and $\log\ n$ are the bit lengths of m and n, respectively. And on finding an inconsistency, the user can detect the cheater by performing a single step of the edit distance computation algorithm. The collision-resistant hash functions are used to allow the server to commit to the large intermediate internal values of the computation using small commitments. And in each verification process of these commitments, the user does $O(\log mn)$ hash computations. Therefore, the overall computation complexity of the user is $O(\log^2(mn))$.

On the server side, each pair of servers runs mn steps of the dynamic program and construct the Merkle Hash Tree for the final result. In each step of the dynamic program, the communication complexity between W_{k1} and W_{k2} is $O(\sigma^2) + O(1)$ due to the computation of γ' and γ'' and the minimum finding protocol. In addition, the computation complexity of constructing the Merkle Hash Tree for the result is $O(mn)$. Therefore, the total computation and communication complexity of each server is $O(\sigma^2 mn)$, which means that the complexity of the server is polynomial in the complexity of evaluating f.

5.2 Security Analysis

Theorem 1. *Suppose that the hash function in use is collision resistant, and the probability that a server honestly executes the protocol is p, then our scheme satisfies the security requirement of correctness against malicious servers with probability 1-$(1-p)^N$, where N is the number of server pairs.*

Proof. Review that in our scheme, the servers construct the Merkle Hash Tree for their respective edit distance matrix M and return the root value MH_{root} to the user. MH_{root} can be viewed as a commitment of M. During the following verification process, the malicious server is unable to change the committed intermediate value of M. Otherwise, if a malicious server tampered the intermediate value but successfully pass the correctness verification, which means that the server generated a fake consistency proof that has the same root value of the Merkle Hash Tree as in the committed edit distance matrix M. Thus, he got a collision in some node along the path to the tampered intermediate value, which contradicts our assumption about the collusion resistant hash function.

In the experiment $\mathbf{Exp}_A^C[\mathcal{RCD}, f, \kappa]$, the ProbGen() oracle and Verify() oracle do not leak useful information for the adversary, as the splitted input sequence λ', μ' are uniformly distributed over the the alphabet set Σ. And we assume that the minimum finding protocol is secure in the two party computation model against malicious adversary, so that the output does not leak any useful information either. In the challenge phrase, after the binary search, U asks W_{11} and W_{12} for the consistency proof for the position n_g and n_b of the result vectors $\hat{m}^{(1)'}$ and $\hat{m}^{(1)''}$. If all the proofs are verified to be valid, U simulates one step of the edit distance computation at the position n_g. The verification equation $M(\alpha,\beta+1) = M^{(1)'}(\alpha,\beta+1) + M^{(1)''}(\alpha,\beta+1)$ holds with probability 1 if the first server is honest and 0 otherwise. For N pairs of servers, the probability that at least one pair of servers are honest is 1-$(1-p)^N$, the probability

is also the probability that the user can get the correct result in our scheme. □

Theorem 2. *Suppose that any pair of servers do not collude with each other, the additively homomorphic encryption scheme is semantically secure, 1-out-of-n OT is receiver secure, and the minimum finding protocol is secure in two-party computation model against malicious adversaries. Then, our scheme satisfies the security requirement of input an output privacy.*

Proof. Firstly, recall that the original sequence λ and μ are splitted into λ',λ'' and μ', μ'', such that $\lambda_i = \lambda'_i + \lambda''_i \mod \sigma$, $\mu_j = \mu'_j + \mu''_j \mod \sigma$. λ'_i, λ''_i, μ'_j and μ''_j are uniformly distributed over the alphabet set $\sum$. Therefore, in the experiment $\mathbf{Exp}_A^{IPriv}[\mathcal{RCD}, f, \lambda]$, the oracle reply of $\mathsf{PubProbGen}(\cdot)$ does not leak any information about the original sequence to the malicious server. Secondly, we assume that the additively homomorphic encryption scheme is semantically secure, which assures that the server W_{k1} does not leak any information about λ' to W_{k2} in the interactive algorithm $\sigma_y \leftarrow \mathsf{Compute}(\sigma_x)$. The OT scheme used in our scheme is receiver secure, so nothing about λ'' is leaked to W_{k1} either. In summarization, our scheme satisfies the security requirement of input privacy. For output privacy, the secure minimum finding protocol used in our protocol guarantees that W_{k1} obtains $M^{(k)'}(i,j)$ but nothing about $M^{(k)''}(i,j)$, meanwhile W_{k2} gets $M^{(k)''}(i,j)$ but nothing about $M^{(k)'}(i,j)$, such that $M(i,j) = M^{(k)'}(i,j) + M^{(k)''}(i,j)$. Therefore, in the challenge phrase, if any adversary is able to corrupt the output privacy our scheme, he can successfully attack the underlying minimum finding protocol with the same probability, which contradicts our assumption. □

6 Conclusion

In this paper, we propose the refereed computation delegation of sequence comparison for the first time. Compared with previous computation delegation schemes, our scheme is qualitatively more practical. Our scheme works in the multi-server model, and the user can get the correct result of the computation as long as there is at least one honest server. For the sequences of length m and n, respectively, our scheme reduce the user computation complexity from $O(mn)$ to $O(\log^2(mn))$. The security analysis shows that our scheme satisfies the security requirements of I/O privacy and correctness.

Acknowledgements

This work is supported by the National Natural Science Foundation of China (No. 61379154, 61100224 and U1135001), the Specialized Research Fund for the Doctoral Program of Higher Education, and Guangdong Natural Science Foundation (No.S2013010013671)

References

[1] B. Applebaum, Y. Ishai, and E. Kushilevitz, "From secrecy to soundness: Efficient verification via secure computation," in *Automata, languages and Programming*, LNCS 6198, pp. 152–163, 2010.

[2] M. J. Atallah and K. B. Frikken, "Securely outsourcing linear algebra computations," in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security (ASIACCS'10)*, pp. 48–59, Beijing, China, Apr. 2010.

[3] M. J. Atallah, F. Kerschbaum, and W. Du, "Secure and private sequence comparisons," in *Proceedings of the 2003 ACM Workshop on Privacy in the Electronic Society (WPES'03)*, pp. 39–44, Washington, DC, USA, Oct. 2003.

[4] M. J. Atallah and J. Li, "Secure outsourcing of sequence comparisons," *International Journal of Information Security*, vol. 4, pp. 277–287, 2005.

[5] M. J. Atallah, K. N. Pantazopoulos, J. R. Rice, and E. E. Spafford, "Secure outsourcing of scientific computations," *Advance in Computers*, vol. 54, pp. 215–272, 2002.

[6] L. Babai, "Trading group theory for randomness," in *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing (STOC'85)*, pp. 421–429, Rhode Island, USA, May 1985.

[7] S. Benabbas, R. Gennaro, and Y. Vahlis, "Verifiable delegation of computation over large datasets," in *Proceedings of the Crypto'11*, pp. 111–131, Santa Barbara, CA, USA, Aug. 2011.

[8] D. Benjamin and M. J. Atallah, "Private and Cheating-free outsourcing of algebraic computations," in *Proceedings of Privacy, Security and Trust (PST'08)*, pp. 240–245, Fredericton, NB, Oct. 2008.

[9] M. Blanton and M. Aliasgari, "Secure outsourcing of DNA searching via finite automata," in *Data and Applications Security and Privacy*, LNCS 6166, pp. 49–64, 2010.

[10] M. Blanton, M. J. Atallah, K. B. Frikken, and Q. Malluhi, "Secure and efficient outsourcing of sequence comparisions," in *Proceedings of the 17th European Symposium on Research in Computer Security (ESORICS'12)*, pp. 505–522, Pisa, Italy, Sep. 2012.

[11] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proceedings of Eurocrypt'04*, pp. 506–522, Interlaken, Switzerland, May 2004.

[12] R. Canetti, B. Riva, and G. N. Rothblum, "Practical delegation of computation using multiple servers," in *Proceedings of the 18th ACM Conference on Computer and Communications Security(ACM CCS'11)*, pp. 445–454, Chicago, USA, Oct. 2011.

[13] P. S. Chu, C. C. Lee, P. S. Chu, P. S. Chung, and M. S. Hwang, "A survey on attribute-based encryption schemes of access control in cloud environments," *Internation Journal of Network Security*, vol. 15, no. 4, pp. 231–240, 2013.

[14] K. M. Chung, Y. Kalai, and S. Vadhan, "Efficient private matching and set intersection," in *Proceedings of the Eurocrypt'04*, pp. 1–19, Interlaken, Switzerland, May 2004.

[15] K. M. Chung, Y. Kalai, and S. Vadhan, "Improved delegation of computation using fully homomorphic encryption," in *Proceedings of the Crypto'10*, pp. 483–501, Santa Barbara, CA, USA, Aug. 2010.

[16] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *Proceedings of the Crypto'10*, pp. 465–482, Barbara, CA, USA, Aug. 2010.

[17] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing (STOC'09)*, pp. 169–178, Bethesda, Maryland, USA, May, June 2009.

[18] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum, "Delegating computation: Interactive proofs for muggles," in *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing (STOC'08)*, pp. 113–122, Victoria, British Columbia, Canada, May 2008.

[19] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM Journal on Computing*, vol. 18, no. 1, pp. 186–208, 1989.

[20] M. Green, S. Honhenberger, and B. Waters, "Outsourcing the decryption of ABE ciphertexts," in *Proceedings of the USENIX Security Symposium*, pp. 34–49, San Francisco, CA, USA, Aug. 2011.

[21] C. Hazay and Y. Lindell, "Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries," in *Proceedings of the Fifth Theory of Cryptography Conference (TCC'08)*, pp. 155–175, New York, USA, Mar. 2008.

[22] C. Hazay and T. Toft, "Computationally secure pattern matching in the presence of malicious adversaries," in *Proceedings of the Asiacrypt'10*, pp. 195–212, Singapore, Dec. 2010.

[23] M. Jakobsson and S. Wetzel, "Secure Server-Aided signature generation," in *Proceedings of 4th International Workshop on Practice and Theory in Public Key Cryptosystems (PKC'01)*, pp. 383–401, Cheju Island, Korea, Feb. 2001.

[24] A. Jarrous and B. Pinkas, "Secure hamming distance based computation and Its applications," in *Proceedings of 7th International Conference (ACNS'09)*, pp. 107–124, Paris-Rocquencourt, France, Jun. 2009.

[25] S. Kawaumura and A. Shimbo, "Fast server-aided secret computation protocols for modular exponentiation," *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 5, pp. 778–784, 2002.

[26] J. Kilian, "A note on efficient Zero-Knowledge proofs and arguments (extended abstract)," in *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, pp. 723–732, Victoria, British Columbia, Canada, May 1992.

[27] J. Kilian, "Improved efficient arguments," in *Proceedings of the 15th Annual International Cryptology Conference (Crypto'95)*, pp. 311–324, Santa Barbara, California, Aug. 1995.

[28] C. C. Lee, S. T. Hsu, and M. S. Hwang, "A study of conjunctive keyword searchable schemes," *Internation Journal of Network Security*, vol. 15, no. 5, pp. 321–330, 2013.

[29] J. Li, X. Chen, J. Li, C. Jia, J. Ma, and W. Lou, "Fine-grained access control based on outsourced attribute-based encryption," in *Proceedings of the 18th European Symposium on Research in Computer Security*, pp. 592–609, Egham, UK, Sep. 2013.

[30] J. Li, X. Huang, J. Li, X. Chen, and X. Yang, "Securely outsourcing attribute-based encryption with checkability," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 8, pp. 2201–2210, 2013.

[31] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Proceedings of CRYPTO'87*, pp. 369–378, Santa Barbara, California, USA, Aug. 1987.

[32] S. Micali, "Computationally sound proofs," *SIAM Journal on Computing*, vol. 30, no. 4, pp. 1253–1298, 2000.

[33] T. Okamoto and S. Uchiyama, "A new Public-Key cryptosystem as secure as factoring," in *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques (Eurcrypt'98)*, pp. 308–318, Espoo, Finland, May, June 1998.

[34] P. Paillier, "Public-Key cryptosystems based on composite degree residuosity classes," in *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques (Eurocrypt'99)*, pp. 223–238, Prague, Czech Republic, May 1999.

[35] B. Parno, M. Raykova, and V. Vaikuntanathan, "How to delegate and verify in public: Verifiable computation from attribute-based encryption," in *Proceedings of the 9th International Conference on Theory of Cryptography (TCC'12)*, pp. 422–439, Taormina, Italy, Mar. 2012.

[36] M. O. Rabin, *How to Exchange Secrets by Oblivious Transfer*, Technical Report TR-81, Mar. 1981.

[37] S. Shankar, *Amazon Elastic Compute Cloud*, Technical Report CS 267, Sept. 2009.

[38] J. R. Troncoso-Pastoriza, S. Katzenbeisser, and M. Celik, "Privacy preserving error resilient DNA searching through oblivious automata," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, pp. 519–528, Alexandria, VA, USA, Oct. 2007.

[39] B. D. Vergnaud, "Efficient and secure generalized pattern matching via fast fourier transformn," in *Proceedings of the 4th International Conference on Cryptology in Africa (Africarypt'11)*, pp. 41–58, Dakar, Senegal, Jule 2011.

[40] R. A. Wagner and M. J. Fischer, "The String-to-String correction problem," *Journal of the ACM*, vol. 21, no. 1, pp. 168–173, 1974.

Xu Ma is a Ph.D. candidate of School of Information Science and Technology, Sun Yat-sen university, China. His research mainly focuses on cryptography and its applications, especially on secure outsourcing computation.

Jin Li received his B.S. (2002) and M.S. (2004) from Southwest University and Sun Yat-sen University, both in Mathematics. He got his Ph.D degree in information security from Sun Yat-sen University at 2007. Currently, he works at Guangzhou University. His research interests include Applied Cryptography and Security in Cloud Computing (secure outsourcing computation and cloud storage).

Fangguo Zhang received his PhD from the School of Communication Engineering, Xidian University in 2001. He is currently a Professor at the School of Information Science and Technology of Sun Yat-sen University, China. He is the co-director of Guangdong Key Laboratory of Information Security Technology. His research mainly focuses on cryptography and its applications. Specific interests are elliptic curve cryptography, secure multiparty computation, anonymity and privacy.