Journal of Web Engineering, Vol. 15, No. 3&4 (2016) 203–225

PRIVACY-PRESERVING COLLABORATIVE WEB SERVICES QOS PREDICTION VIA YAO'S GARBLED CIRCUITS AND HOMOMORPHIC ENCRYPTION

LU LI[#1], AN LIU[*2], QING LI[†3], GUANFENG LIU[*4], ZHIXU LI[*5]

[#] *School of Information Engineering, Yancheng Teachers University, China*
[1] *liluzq@mail.ustc.edu.cn*

[*] *School of Computer Science and Technology, Soochow University, China*
[2,4,5] {*anliu, gfliu, zhixuli*}*@suda.edu.cn*

[*] *Department of Computer Science, City University of Hong Kong, China*
[3] *itqli@cityu.edu.hk*

Received December 10, 2014
Revised November 30, 2015

Collaborative Web services QoS prediction has become an important tool for the generation of accurate personalized QoS which is a cornerstone of most QoS-based approaches for Web services selection and composition. While a number of achievements have been attained on the study of improving the accuracy of collaborative QoS prediction, little work has been done for protecting user privacy in this process. In this paper, we propose a privacy-preserving collaborative QoS prediction framework which can protect the private data of users while retaining the ability of generating accurate QoS prediction. We combine Yao's garbled circuit and additively homomorphic encryption via additively secret sharing to address non-linear computations required in the process of QoS prediction. We implement the proposed framework based on FasterGC, an open source implementation of Yao's garbled circuit, and conduct extensive simulations to study its performance. Simulation results, together with theoretical security and complexity analysis, show that privacy-preserving QoS prediction can be efficiently achieved in our framework.

Keywords: collaborative QoS prediction, privacy-preserving, Yao's garbled circuits, homomorphic encryption, recommendation system
Communicated by: D. Schwabe & T. Tokuda

1. Introduction

Web services, which are Internet based programmable application components, have made great progress in the past decade. At its early stage, Web services technology was mainly exploited to build traditional software applications such as online reservation and stock trading. Along with the rapid development of cloud computing, more and more corporations including Amazon and Google offer cloud Web services, such as Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS), for the realization of scalable and cost-effective computing [15]. Moreover, numerous real-world devices can also provide Web services through Device Profile for Web Services (DPWS) or RESTful API [17],

which significantly increases not only opportunities but also challenges of Service Oriented Computing (SOC).

QoS-based Web services selection and composition has been discussed extensively in the recent literature [1, 2, 3, 42, 43]. A common assumption of these proposed approaches is that accurate QoS values of Web services are always available. It is, however, still an open problem to obtain accurate QoS values. On one hand, the QoS values advertised by service providers or third-party communities are not accurate to service users, as they are susceptible to the uncertain Internet environment and user context [7]. On the other hand, it is impractical for service users to directly evaluate the QoS of all available services due to the constraints of time, cost and other resources [48]. As an effective solution to this problem, personalized collaborative Web services QoS prediction has received much attention recently [21, 35, 44, 46]. The basic idea is that similar users tend to observe similar QoS for the same service, so it is possible to predict the QoS value of a service observed by a user based on the QoS values of the service observed by the similar users to this particular user. By this kind of computation, different users are typically given different QoS prediction values even for the same service and the final prediction values in fact depends on their specific context. To enable collaborative QoS prediction to work well, users must provide their observed QoS values of the services they have invoked. Based on these provided QoS values, a variety of techniques have been employed to improve the quality especially accuracy of prediction [7, 38, 40, 41, 47, 48].

Though many achievements have been attained on the study of improving the accuracy of collaborative QoS prediction, little work has been done for protecting user privacy in this process. In fact, the observed QoS values could be a sensitive information, so users may not be willing to share them with others. For example, the observed response time reported by a user typically depends on her location [7, 38], which means that the user's location could be deduced from the QoS information she provided. Consequently, an interesting but challenging question is whether or not a recommendation system can make accurately personalized QoS prediction for users while respecting their privacy.

In this paper, we provide a positive answer to the above question. More specifically, we propose a privacy-preserving QoS prediction framework which can protect the privacy of the users by means of encrypting their private data, that is, their observed QoS, and meanwhile can make an accurate QoS prediction in the encrypted domain by running cryptographic protocols. It should be noted that the difficulty of implementing such a framework depends on the trade off between efficiency and accuracy. Generally, complicated computation is required to improve the prediction accuracy, but it is inefficient even impractical to apply these computation in the encrypted domain. On the other hand, while realizing an efficient privacy-preserving QoS prediction is possible, the prediction accuracy may not be satisfied. By keeping this trade off in mind, we select an effective collaborative QoS prediction approach proposed by Zheng *et al.* in [46] which combines user-based collaborative filtering and item-based collaborative filtering to achieve better QoS prediction accuracy. As will be discussed in detail in later sections, Zheng's approach contains a number of nonlinear computations, which makes it impractical to construct a privacy-preserving recommendation system totally based on homomorphic encryption, just as the one presented in [13]. To address this problem, we combine Yao's garbled circuit [39] and additively homomorphic encryption [32, 10] via additively secret sharing. The main contributions of this work are two-fold:

- A complete privacy-preserving collaborative Web services QoS prediction framework is designed to bridge the gap between personalization and privacy. The prediction accuracy is guaranteed by Zheng's approach, while the accompanied complex nonlinear computations are addressed efficiently by the delicate combination of Yao's garbled circuit and additively homomorphic encryption.
- The privacy-preserving collaborative QoS prediction framework is implemented based on an open source framework FasterGC [19]. Various optimization techniques adopted in FasterGC enables the framework to realize quite efficient QoS prediction, thus making privacy-preserving QoS prediction not only theoretical interesting but also practical in real applications.

The remainder of this paper is organized as follows: section introduces a representative collaborative Web services QoS prediction framework and briefly reviews some cryptology techniques used to building our privacy-preserving solution. Section presents the system architecture of our privacy-preserving QoS prediction framework. In Section , we present the details of our approach, including the design of Yao's garbled circuits for similarity calculation and QoS prediction, and the secure Top-K query. Security analysis and simulation results of the proposed framework are presented in Section and Section , respectively. Finally, Section discusses related work and Section concludes the paper.

2. Preliminaries

In this section, we first briefly introduce a representative approach for collaborative Web services QoS prediction. Then, we review some cryptology background foundations of our privacy-preserving QoS prediction framework.

2.1. *Collaborative Web Services QoS Prediction*

In the collaborative Web services QoS prediction, a user is required to provide the observed QoS of the services her has invoked to the recommendation system. Based on the collected QoS values, the recommendation system can predict the QoS of all available services for a user through some complicated algorithms. The more service QoS values her provides, the higher prediction accuracy can be achieved as more user features can be mined from the provided data. In [47], the authors present a typical framework of collaborative QoS prediction, which consists of three primary modules. The first module is in charge of calculating the similarity between any two users (or services) based on the QoS values provided by users. After that, for a given user (or item), a set of similar users (or items) can be identified by the second module based on the similarity values. Finally, the third module generates the predicted QoS of all available services by exploiting past service usage experiences of similar users. In practice, a user can provide her observed QoS to the recommendation system anytime and can obtain the predicted QoS of the services she is interested in anytime.

To realize the abstract functionality of the above three modules, some specific algorithms must be designed. Here, we introduce an effective approach proposed in [46] which is also served as the foundation of our work. In this approach, two types of similarity are calculated in order to improve prediction accuracy: user similarity and service similarity. In particular, the similarity between two users u_i and $u_{i'}$ are calculated based on the services they commonly

invoked using the following equations:

$$Sim(u_i, u_{i'}) = \frac{\omega_{i,i'} \sum_{s_j \in S}(q_{i,j} - \bar{q}_i)(q_{i',j} - \bar{q}_{i'})}{\sqrt{\sum_{s_j \in S}(q_{i,j} - \bar{q}_i)^2 \sum_{s_j \in S}(q_{i',j} - \bar{q}_{i'})^2}}, \tag{1}$$

$$\omega_{i,i'} = \frac{2 \times |S_i \bigcap S_{i'}|}{|S_i| + |S_{i'}|}, \tag{2}$$

where $S = S_i \bigcap S_{i'}$ is the set of services that user u_i and user $u_{i'}$ commonly invoked, $r_{i,j}$ is the QoS value of service j observed by user i, $\bar{q}_i$ is the average QoS value of all services observed by user u_i, and $\omega_{i,i'}$ is a weight to devalue the similarity between two users if they are actually not similar but happen to have similar QoS experience on a few co-invoked services.

Likewise, the similarity between two services s_j and $s_{j'}$ are calculated as follows:

$$Sim(s_j, s_{j'}) = \frac{\omega_{j,j'} \sum_{u_i \in U}(q_{i,j} - \bar{q}_j)(q_{i,j'} - \bar{q}_{j'})}{\sqrt{\sum_{u_i \in U}(q_{i,j} - \bar{q}_j)^2 \sum_{u_i \in U}(q_{i,j'} - \bar{q}_{j'})^2}}, \tag{3}$$

$$\omega_{j,j'} = \frac{2 \times |U_j \bigcap U_{j'}|}{|U_j| + |U_{j'}|}, \tag{4}$$

where $U = U_j \bigcap U_{j'}$ is the set of users who invoke both service j and service j', $\bar{q}_j$ is the average QoS value of service j observed by different users, and $\omega_{j,j'}$ is also a devaluation weight to improve prediction accuracy.

Note that that both $Sim(u_i, u_{i'})$ and $Sim(s_j, s_{j'})$ are ranging from [-1, 1], and a larger value indicates that two users (or services) are more similar [46].

Based on the above similarity values, similar users can be identified by

$$SU(u_i) = \{u_{i'} | u_{i'} \in T(u_i), Sim(u_i, u_{i'}) > 0, u_i \neq u_{i'}\}, \tag{5}$$

and similar services can be identified by

$$SS(s_j) = \{s_{j'} | s_{j'} \in T(s_j), Sim(s_j, s_{j'}) > 0, s_j \neq s_{j'}\}, \tag{6}$$

where $T(u_i)$ is a set of the Top-K similar users to user u_i, $T(s_j)$ is a set of the Top-K similar services to service s_j, and $Sim(u, u_i) > 0$ (or $Sim(s, s_j) > 0$)) excludes the dissimilar users (or services) with negative similarity values.

To predict the QoS value of service s_j observed by user u_i, one way is to make use of the similar users to user u_i through the following equation:

$$P_u(q_{i,j}) = \bar{q}_i + \sum_{u_{i'} \in SU(u_i)} \frac{Sim(u_i, u_{i'})(q_{i',j} - \bar{q}_{i'})}{\sum_{u_{i'} \in SU(u_i)} Sim(u_i, u_{i'})}, \tag{7}$$

and the other way is to make use of the similar services to service s_j as follows:

$$P_s(q_{i,j}) = \bar{q}_j + \sum_{s_{j'} \in SS(s_j)} \frac{Sim(s_j, s_{j'})(q_{i,j'} - \bar{q}_{j'})}{\sum_{s_{j'} \in SS(s_j)} Sim(s_j, s_{j'})}. \tag{8}$$

As proved in [46], these two ways can be combined together to improve the accuracy of QoS prediction. Specifically, the final predicted value $P(q_{i,j})$ is a weighted sum of $P_u(q_{i,j})$ and $P_s(q_{i,j})$ defined by:

$$P(q_{i,j}) = \omega_u P_u(q_{i,j}) + \omega_s P_s(q_{i,j}), \tag{9}$$

$$\omega_u = \frac{\lambda c_u}{\lambda c_u + (1-\lambda)c_s}, \tag{10}$$

$$\omega_s = \frac{(1-\lambda)c_s}{\lambda c_u + (1-\lambda)c_s}, \tag{11}$$

where λ is an adjustable parameter to control the contribution of the two prediction ways to the final predicted QoS value, c_u and c_s are confidence weights defined as follows:

$$c_u = \sum_{u_{i'} \in SU(u_i)} \frac{Sim(u_i, u_{i'}) Sim(u_i, u_{i'})}{\sum_{u_{i'} \in SU(u_i)} Sim(u_i, u_{i'})}, \tag{12}$$

$$c_s = \sum_{s_{j'} \in SS(s_j)} \frac{Sim(s_j, s_{j'}) Sim(s_j, s_{j'})}{\sum_{s_{j'} \in SS(s_j)} Sim(s_j, s_{j'})}. \tag{13}$$

2.2. *Cryptology Background*

2.2.1. *Security Definition Under Semi-honest Models*

In this paper, we assume that all operations are done under the semi-honest model, which means that everyone follows a pre-defined protocol, but may use the results and intermediate knowledge that her can obtain during the execution of the protocol to deduce additional information of others' data. According to [4, 16], we give a formal definition of security as follows.

Consider an ideally privacy-preserving implementation of a recommendation algorithm, in which a trusted third party $\mathcal{T}$ runs the algorithm on the inputs of all users to yield the outputs for the users. After that, $\mathcal{T}$ sends the corresponding output to each user and deletes all the data. Clearly, one cannot devise a more private protocol. Now a secure multiparty protocol $\mathbb{P}$ for the algorithm is private if any attack against $\mathbb{P}$ can be converted to an attack against ideal implementation, and these two attacks have roughly the same success rate and the same time-complexity. Given a protocol $\mathbb{P}$, if all intermediate messages sent and received by a corrupted party can be simulated efficiently based on the input and output of $\mathbb{P}$, then $\mathbb{P}$ is secure.

2.2.2. *Additively Secret Sharing*

A value x is additively secret shared between two parties $\mathcal{A}$ and $\mathcal{B}$ if $\mathcal{A}$ holds a random number x_1 sampled from a sufficiently large domain, $\mathcal{B}$ holds x_2, and $x_1 + x_2 = x$. For brevity, hereafter we let $[\![x]\!]$ denote additive secret sharing of x.

2.2.3. *Yao's Garbled Circuits*

Yao's protocol [39, 27] (a.k.a garbled circuits) allows two semi-honest parties respectively holding inputs x and y, to evaluate an arbitrary function $f(x, y)$ without leaking any information about their inputs beyond what can be deduced by the function output. The basic idea is that one party (the garbled-circuit *constructor*) constructs a garbled version of a circuit to compute f, while the other party (the garbled-circuit *evaluator*) then obliviously computes the output of the circuit without learning any intermediate values.

The protocol starts with a boolean circuit evaluating f. To each wire w_i of the circuit, the *constructor* associates two random cryptographic keys $k_{w_i}^0$ and $k_{w_i}^1$, which are respectively corresponded to the bit-values $b_i = 0$ and $b_i = 1$. For each binary gate g, with input wires (w_i, w_j) and output wire w_k, the constructor first computes four values

$$E_{(k_{w_i}^{b_i}, k_{w_j}^{b_j})} k_{w_k}^{g(b_i,b_j)}, b_i, b_j \in \{0, 1\}, \tag{14}$$

and then sends them to the evaluator party in random orders. Given the pair of keys corresponding to input wires of a binary gate, the *evaluator* can recover the key of the output wire by decrypting the values. It is worth noting that only the value $k_{w_k}^{g(b_i,b_j)}$ can be obtained, and that no other output values can be recovered for the corresponding gate. For a given function $f(x, y)$, the parties first compile it to a boolean circuit. Then, the *constructor* garbles the circuit by providing each gate four ciphertexts. These values are then send to the *evaluator*. The only problem is how the *evaluator* obtains the corresponding keys to each input wire of the circuit. The real inputs are holding by two parties independently. To the values holding by the *constructor*, she can directly send the appropriate keys to the *evaluator*. To the values holding by the *evaluator*, a cryptology tool called *1-out-of-2* oblivious transfer [34, 14] can be used to enable the *evaluator* to obtain the desired keys without leaking her private information. Once obtaining the keys corresponding to all the input wires, the *evaluator* can locally compute the garbled circuit gate-by-gate. This protocol also enables the *evaluator* to obtain the final output by simply modifying the output values using the real bit-values.

Beginning with [28], there are many efficient implementations of Yao's protocol [6, 18, 26, 33, 19, 25]. In [5], the authors present a very good overview of garbled circuits. These efforts are good candidates serving as subroutine for large privacy-preserving protocols. In fact, most subroutines require shared output. In our framework, we need the output additively shared between two parties. Assume both parties want to use Yao's protocol to obtain $[\![f(x, y)]\!]$. We use a simple method to do this. Let the *constructor* first generates a random number r from a sufficiently large domain, then she constructs a boolean circuit for calculating $f'(x, y)$, such that $f'(x, y) = f(x, y) - r$. She uses r as output, and the other party uses the calculating result $f'(x, y)$ as output.

In our framework, Yao's garbled circuit is implemented based on FasterGC [19], which is a Java-based open-source framework that enables developers to compute arbitrary circuits using elementary *XOR*, *OR* and *AND* gates. This framework includes several optimizations. First, the communication and computation cost for XOR gates in the circuits is significantly reduced using the "free XOR" techniques [23]. Second, it gives a 1/4 communication saving for the communication cost of 2-fan-in non-XOR gates by using the garbled-row reduction technique [33]. Third, FasterGC implements the Oblivious Transfer (OT) extension [20] which can execute a practically unlimited number of transfers at the cost of k OTs where k is a (statistical) security parameter, and several symmetric-key operations per additional OT. Fi-

nally, FasterGC contains many improved computing modules designed by [24] to use as many as "free XOR" gates. These optimizations enable FasterGC to be the fastest implementation of Yao's garbled circuit currently.

2.2.4. *Additively Homomorphic Encryption*

Given a key pair (E, D) and a message m in Z_N^*, $e = E(m)$ denotes an encryption of the plaintext m, and $d = D(e)$ denotes the decryption of the ciphertext e. A cryptosystem is additively homomorphic if $D(E(m_1) \times E(m_2)) = m_1 + m_2$ and there exists an efficient re-randomization procedure that takes in a valid encryption $E(m)$ and outputs a random encryption of m such that even an adversary with infinite power cannot deduce how the message m was computed. There are many practical additively homomorphic encryption systems, such as Paillier [32] and DGK [10]. In this paper, we use the Paillier cryptosystem to encrypt the input data of the users and to construct a cryptology protocol to query the Top-K users. The main idea behind Paillier scheme is given below:

The encryption of a message, $m \in Z_N$, by using Paillier scheme is defined as

$$E(m, r) = g^m \times r^N mod(N^2), \tag{15}$$

where N is a product of two large prime numbers p and q, g generates a subgroup of order N, and r is a random number in Z_N^*. The public key is (N, g) and the private key is (p, q). The additively property of the Paillier cryptosystem can be easily verified as shown below:

$$\begin{aligned} E(m_1, r_1) \times E(m_2, r_2) =& (g_1^m \times r_1^N) \times (g_1^m \times r_1^N) mod(N^2) \\ =& g^{m_1+m_2} \times (r_1 * r_2)^N mod(N^2) \\ =& E(m_1 + m_2, r_1 \times r_2). \end{aligned}$$

Yao's garbled circuit and homomorphic encryption are both effective tools for constructing secure protocols. Considering the characteristic of collaborative QoS prediction, we design a privacy-preserving version by combining Yao's garbled circuit and additively homomorphic encryption via additively secret sharing.

2.2.5. *Example*

We introduce a simple example here to illustrate the usage of the above cryptology techniques. Suppose that we want to calculate a simple function $f(d_1, d_2) = d_1 d_2$ without leaking d_1 and d_2 to anyone. A solution based on the techniques discussed above is as follows. First, we assume there are two roles, constructor and evaluator, in the subsequent computation. Then, both d_1 and d_2 are additively secret shared between the constructor and the evaluator as follows:

- d_1 and d_2 are encrypted by the public key of the evaluator and given to the constructor;
- the constructor randomly generates two numbers r_1 and r_2, computes $E(d_1) \times E(r_1)$ and $E(d_2) \times E(r_2)$, and sends the corresponding results $E(d_1 + r_1)$ and $E(d_2 + r_2)$ to the evaluator;
- the evaluator decrypts the ciphertext by her private key and obtains the plaintext $d_1 + r_1$ and $d_2 + r_2$;

- now, the constructor holds $-r_1$ and $-r_2$, and the evaluator holds d_1+r_1 and d_2+r_2, so d_1 and d_2 are both additively secret shared as $-r_1+(d_1+r_1) = d_1$ and $-r_2+(d_2+r_2) = d_2$.

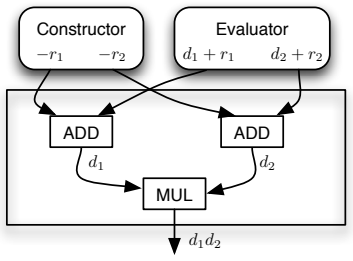

Fig. 1. A simple circuit for secure multiplication

After that, the constructor and the evaluator cooperatively construct a garbled circuit with respective inputs $\langle -x_1, -x_2 \rangle$ and $\langle d_1 + x_1, d_2 + x_2 \rangle$. Fig.1 is a conceptual view of this circuit, which consists of three basic circuits: two *ADD* circuits to realize the additive of two numbers and one *MUL* circuit to realize the multiplication of two numbers. Each *ADD* circuit takes as input one data from the constructor (e.g., $-x_1$) and one data form the evaluator (e.g., $d_1 + x_1$), and generates the corresponding sum which is encoded as part of the garbled-circuit computation (e.g., $g(d_1)$, though in the figure we still use d_1). Then, the *MUL* circuit takes the outputs of the two *ADD* circuits as its input and generates the corresponding product. Note that the final result of a garbled circuit could be in the form of inner encoding (e.g., $g(d_1 d_2)$) which can be handled by other circuits, or in the form of plaintext (e.g., $d_1 d_2$) so that it can be understood by other applications. Note also that the final result is hold by the evaluator according to the realization of garbled circuits.

3. System Architecture

This section gives an overview of our approach which actually can be regarded as a privacy-preserving version of the collaborative QoS prediction framework described in Section . As shown in Fig. 2, there are three roles in our framework:

- *Users*: Users are the consumers of various Web services including the crypto service mentioned below. They are required to provide the observed QoS values of services they have invoked to a recommendation system, and, as a reward, they can obtain recommendations especially QoS prediction of unknown Web services from the recommendation system.

- *Recommendation System (RS)*: RS has a business interest in generating recommendations especially QoS prediction for the users.

- *Crypto Service Provider (CSP)*: CSP is a third party who has a business interest in providing cryptographic service. She has private keys for the Paillier cryptosystem.

All the above roles are assumed to be semi-honest. The goal of our framework is to make QoS prediction for the users without leaking any piece of the private information about

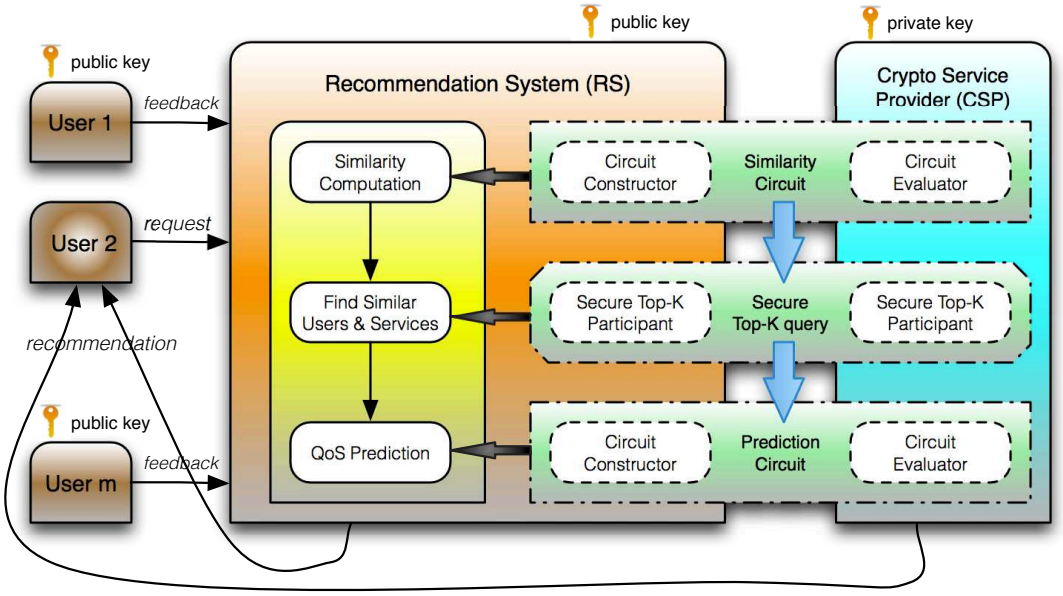

Fig. 2. System architecture of privacy-preserving collaborative QoS prediction

them. As a preliminary step of privacy-preserving collaborative QoS prediction, users need to encrypt their observed QoS values by the public key of CSP before sending them to RS. More specifically, if user u_i has invoked service s_j and observed a QoS value $q_{i,j}$, she needs to send the encrypted values $E(q_{i,j})$ and $E(-q_{i,j})$ to RS. Besides, she needs to send RS $E(-\bar{q}_i)$ where $\bar{q}_i$ is the average observed QoS value of all services she has invoked. Recall that RS does not have the corresponding decryption key, so RS cannot access the sensitive data of user u_i.

As mentioned in Section , a typical recommendation system for QoS prediction has three modules: similarity computation, find similar users and services, and QoS prediction. In our framework, every module is endowed with the ability of privacy-preserving through the combination of Yao's garbled circuits and homomorphic encryption. More specifically, given a user u who wants to know a predicted QoS of service s, RS and CSP will cooperatively construct a garbled circuit which can compute the similarity values between u and other users as well as the similarity values between s and other services based on the encrypted data provided by users. The outputs of this garbled circuit, that is, similarity values, are given to the secure Top-K protocol cooperatively run by RS and CSP to generate the Top-K similar users to u and Top-K similar services to s. The secure Top-K are implemented based on homomorphic encryption and a special garbled circuit called *comparison* circuit. Based on these similar users and similar services, RS and CSP will cooperatively construct again a garbled circuit to calculate the expected QoS of s observed by u. All the intermediate results are in the form of either ciphertext or the inner encoding of garbled-circuit computation, and therefore are kept secret to both RS and CSP.

Theoretically, the above computation can be performed after RS receives a query from

a user. In practice, however, we notice that this computation can be carried out during the idle time of RS and PSP even before any user asks for prediction. That is, though the privacy-preserving version is somewhat time-consuming especially compared with the original version of collaborative QoS prediction, a user will receive prediction from RS shortly after her request without any delays.

4. Privacy-Preserving QoS Prediction

In this section, we present the specific design of our privacy-preserving QoS prediction framework. Section introduces the details of garbled circuits for the calculation of user similarity and service similarity. Section explains the secure Top-K query which can find Top-K similar users and services in a encrypted domain, and finally Section gives the garbled circuits for the final QoS prediction.

4.1. *Similarity Calculation*

4.1.1. *Circuit Composition*

The similarity calculation through formula 1 and formula 3 requires the combination of multiple basic operations including additive, multiplication, division and square root. The latest version of FasterGC library does not support multiplication, division and square root, so we first extend it by providing several modules to implement these operations. In [30], the authors present a good overview of implementation of these modules. Multiplication can be implemented by using the Karatsuba algorithm [22], where the cost of multiplying two k-bit is at most $3k^{log_2 3}$ single digit operations. Considering that k is very short in the context of QoS prediction, we implement the multiplication using standard school method, which costs k^2 bit operations. The division is also implemented by standard school method. The square root implementation is realized using an iterative but data-agnostic method presented in [9].

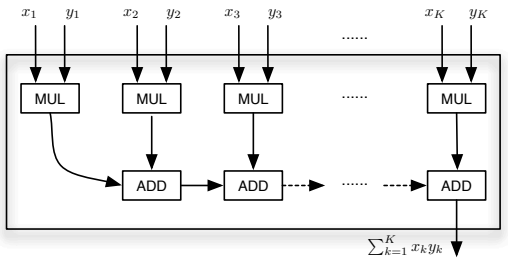

Fig. 3. Conceptual view of *SoP* circuit. Its input is two vectors $X = [x_1, x_2, \cdots, x_K]$ and $Y = [y_1, y_2, \cdots, y_K]$. Its output is their dot product $\sum_{k=1}^{K} x_k y_k$

Based on these basic operations, we can construct a composite circuit to calculate similarity. Note that, formula 1 and formula 3 have the following common structure:

$$\frac{\omega \sum XY}{\sqrt{\sum X^2 \sum Y^2}} \tag{16}$$

where $X = \{x_1, x_2, \cdots, x_K\}$ and $Y = \{y_1, y_2, \cdots, y_K\}$ are two vectors and ω is a real number. As $\sum XY$ is an important kind of computation here, we first construct a composite circuit *SoP* (Sum of Product). Fig. 3 shows the conceptual view of a *SoP* circuit which consists of K *MUL* circuits and $K-1$ *ADD* circuits. The k^{th} *MUL* circuit generates $x_k y_k$ by taking x_k and y_k as its input. The k^{th} *ADD* circuit produces the sum of the output of its directly previous *ADD* circuit and the output of k^{th} *MUL* circuit. It is easy to verify that the output of the last *ADD* circuit is $\sum_{k=1}^{K} x_k y_k$.

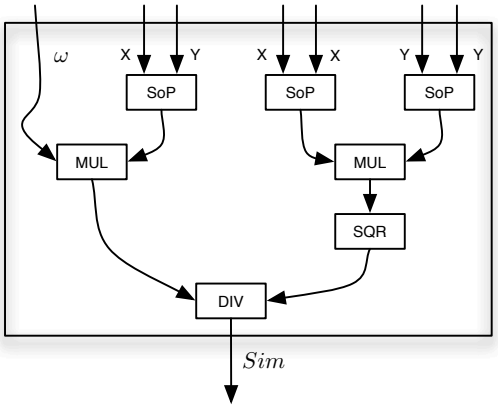

Fig. 4. Conceptual view of *SIM* circuit. Its input is one real number ω and two vectors X and Y. Its output is $\frac{\omega \sum XY}{\sqrt{\sum X^2 \sum Y^2}}$

The composite circuit *SIM* for similarity computation is shown in Fig. 4. The numerator in formula 16 is the output of a *MUL* circuit whose input is the real number ω and the output of a *SoP* circuit that computes $\sum XY$. The denominator in formula 16 is generated by a *SQR* circuit which computes the square root of the output of a *MUL* circuit that multiplies $\sum XX$ and $\sum YY$ generated by two *SoP* circuits. By assigning proper values to X and Y, the *SIM* circuit can calculate the similarity defined in formula 1 and formula 3.

Before discussing the assignment of X and Y, we note that Yao's garbled circuits only support computation for integers, so it may cause severe loss of accuracy when conducting division or square root. To overcome this shortcoming, we use fixed-point representation of floats with a λ-bit fractional part where the integer d represents a float $2^{-\lambda}d$. In particular, we have:

1. Division: $a/b = a \cdot 2^{\lambda}/b$
2. Square root: $\sqrt{a} = \sqrt{a \cdot 2^{2\lambda}}$

4.1.2. *Input Preparation*

If privacy issue does not need to be taken into account, the assignment of X and Y for the computation of $Sim(u_i, u_{i'})$ would be quite easy by letting x_j be $q_{i,j} - \bar{q}_i$, y_j be $q_{i',j} - \bar{q}_{i'}$ and K be $|S|$. However, neither RS nor CSP can completely hold $q_{i,j} - \bar{q}_i$ (or $q_{i',j} - \bar{q}_{i'}$) by

herself as this will lead to information leakage about the sensitive data of users. It is therefore necessary to make $q_{i,j} - \bar{q}_i$ and $q_{i',j} - \bar{q}_{i'}$ be additively secret shared between RS and CSP.

The additively secret sharing of $q_{i,j} - \bar{q}_i$ can be achieved as follows. As RS knows $E(q_{i,j})$ and $E(-\bar{q}_i)$, she can compute $E(q_{i,j} - \bar{q}_i)$ which equals to $E(q_{i,j}) \times E(-\bar{q}_i)$. Then, RS randomly generates a number r_j, and sends $E(q_{i,j} - \bar{q}_i + r_j)$ to CSP who can decrypt this value using her private key. Then, as shown in Fig. 5, RS and CSP can cooperatively construct for each service $s_j \in S$ an *ADD* circuit with respective inputs $-r_j$ and $q_{i,j} - \bar{q}_i + r_j$. The output of this circuit is $g(q_{i,j} - \bar{q}_i)$, that is, the inner encoding of $q_{i,j} - \bar{q}_i$ in the garbled-circuit computation, which can be safely assigned to x_j. Note that the purpose of using the random number r_j is to mask the private value $q_{i,j} - \bar{q}_i$, so this random number must be sampled from a sufficiently large domain to achieve statistical security. In general, for an l-bit number, choosing r as a random $(l + \sigma)$-bit integer suffices to give statistical security $O(n * 2^{-\sigma})$. Likewise, $q_{i',j} - \bar{q}_{i'}$ can be additively secret shared and $g(q_{i',j} - \bar{q}_{i'})$ can be safely assigned to y_j.

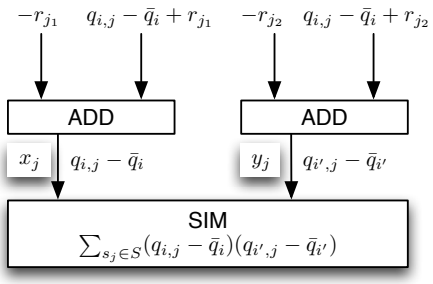

Fig. 5. Procedure of generating input for the *SIM* circuit to compute user similarity

Though user similarity and service similarity have quite similar definitions (see formula 1 and 3), it is much more difficult to realize the additively secret sharing of $q_{i,j} - \bar{q}_j$ for each user $u_i \in U$ as no one knows the values of $\bar{q}_j$. To address this problem, we first note that:

$$\bar{q}_j = \frac{\sum_{u_k \in U_j} q_{k,j}}{|U_j|}, \tag{17}$$

by which $q_{i,j} - \bar{q}_j$ can be rewritten as follows:

$$q_{i,j} - \bar{q}_j = \frac{q_{i,j}|U_j| - \sum_{u_k \in U_j} q_{k,j}}{|U_j|}. \tag{18}$$

Also note that RS holds $|U_j|$, $E(q_{i,j})$, and $E(-q_{k,j})$ for every user $u_k \in U_j$, based on which RS can calculate by herself the encrypted value of the numerator:

$$E(q_{i,j}|U_j| - \sum_{u_k \in U_j} q_{k,j}) = E(q_{i,j})^{|U_j|-1} \prod_{\substack{u_k \in U_j \\ k \neq i}} E(-q_{k,j}) \tag{19}$$

Through the above operation, RS now holds the ciphertext and plaintext of the numerator and the denominator in formula 18, respectively. Then, RS generates a random number r_i in a sufficiently large domain and sends $E(q_{i,j}|U_{s_j}| - \sum_{u_k \in U_{s_j}} q_{k,j} + r_i)$ to CSP who can

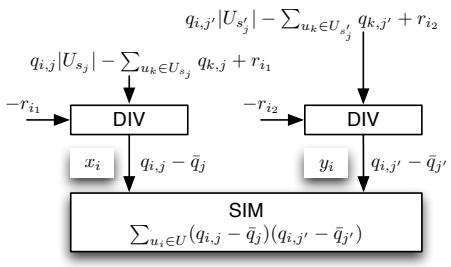

Fig. 6. Procedure of generating input for the *SIM* circuit to compute service similarity

also decrypt this value using her private key. Finally, as shown in Fig. 6, RS and CSP cooperatively construct for each user $u_i \in U$ a *DIV* circuit with respective inputs $-r_i$ and $q_{i,j}|U_{s_j}| - \sum_{u_k \in U_{s_j}} q_{k,j} + r_i$. The output of this circuit is in the form of $g(q_{i,j} - \bar{q}_j)$ and can be safely assigned to x_i. A similar procedure can be applied to the calculation of $q_{i,j'} - \bar{q}_{j'}$ and y_i can be set accordingly.

4.1.3. *Output Processing*

As shown in Fig. 4, the output of the *SIM* circuit is a similarity value, which is also a kind of sensitive information of users and thus cannot be revealed to neither RS nor CSP. Note again that its inner encoding $g(Sim)$ is a part of garbled-circuit computation and is therefore privacy-preserving. In order to facilitate later secure Top-K query, however, we also make it additively secure shared between RS and CSP. For each similarity value Sim_k, RS provides a random number r_k generated from a sufficient large domain. An *ADD* circuit is then created, which takes r_k and $g(Sim_k)$ as input and generates $Sim'_k = r_k + Sim_k$. Consequently, RS holds $-r_k$ and CSP holds Sim'_k, and these two values are also called two additive shares of Sim_k.

4.2. *Secure Top-K Query*

To generate the final prediction, the similarity values between a specific user (or service) and the Top-K similar users (or services) should be calculated. Recall that the similarity values have been shared between RS and CSP, so the problem is equivalent to the following. Two participants RS and CSP additively share an n-dimensional vector $(\llbracket a_1 \rrbracket, \cdots, \llbracket a_n \rrbracket)$ and want to obtain the Top-K values $(\llbracket a_{t_1} \rrbracket, \cdots, \llbracket a_{t_k} \rrbracket)$. A naive solution is to directly use the *Comparison* circuit [19] to compare the elements in the input vector, and then select the desired elements. This process seems to be secure, but in fact it is not. When the parties invoke the *Comparison* circuit, the comparison result will reveal to both parties, and this will lead to substantial information leakage about users. To deal with this problem, we first let two participants engage a shuffle protocol on input vector, and the output of this protocol is a new n-dimensional vector $(\llbracket a_{\pi_1} \rrbracket, \cdots, \llbracket a_{\pi_n} \rrbracket)$, where $\pi(-)$ is a random permutation key where neither of the participants knows it. Suppose that a_i^1 denotes the part of a_i hold by RS and a_i^2 is hold by CSP. The shuffle protocol is given as follows.

1. CSP generates a public-private keypair (E, D) for a homomorphic encryption scheme. CSP encrypts her shares of $(a_1, \cdots, a_n)$ to generate the encrypted sequence $W = (w_1, ..., w_n)$, $w_i = E(a_i^2)$, and then she sends RS the sequence W and the public key E.

2. RS generates a random sequence $R = (r_1, ..., r_n)$ over a sufficient large domain and a random permutation π_1 of n numbers. She encrypts her sequence R to obtain $U = (u_1, ..., u_n)$, $u_i = E(r_i)$, and then she multiplies the components of the sequences W and U to obtain $V = (v_1, ..., v_n)$, where $v_i = w_i * u_i$. RS computes $V' = (v_{\pi_1(1)}, ..., v_{\pi_1(n)})$ and sends it to CSP.

3. CSP decrypts the components of V' to get $(a^2_{\pi_1(1)} + r_{\pi_1(1)}, ..., a^2_{\pi_1(n)} + r_{\pi_1(n)})$.

4. RS outputs $(a^1_{\pi_1(1)} - r_{\pi_1(1)}, ..., a^1_{\pi_1(n)} - r_{\pi_1(n)})$.

5. RS and CSP exchange the roles, and execute the above process again.

By executing this protocol, neither of the parties knows the order of the new sequence. Thus, revealing which elements in the new sequence $(\llbracket a_{\pi(1)} \rrbracket, \cdots, \llbracket a_{\pi(n)} \rrbracket)$ have larger values will not leak any sensitive information. Now the secure Top-K query problem can be converted to the classic Top-K query in which the data is hold by one party, as we can use a *Comparison* circuit to replace comparison operations. In [8], two efficient solutions with $O(n)$ computation complexity can be used to search the K-th ranked element, and we can directly use any of them with post comparing each element with the K-th ranked element to finish the Top-K query. In addition, a simply method which required K rounds query the largest element can also be applied when K is small enough. Once the query process finishes, two participants can easily select the shares $Sim_{u_i,u_{i'}}$, where user $u_{i'}$ is one of the Top-K similar user to u_i.

As seen in formula 7, the computation of $P_u(q_{i,j})$ also requires the value of $q_{i',j} - \bar{q}_{i'}$ for each user $u_{i'} \in SU(u_i)$. This also should be addressed in the secure Top-K query because we cannot identify which services belong to $SU(u_i)$ only based on similarity values and thus cannot determine the value of $q_{i',j} - \bar{q}_{i'}$ after secure Top-K query. We present here an efficient solution based on data binding. First recall that $q_{i',j} - \bar{q}_{i'}$ has already been additively secret shared between RS and CSP after the computation of formula 1 (see Fig. 5), that is, RS holds $r_{i'}$ and CSP holds $q_{i',j} - \bar{q}_{i'} + r_{i'}$. Then, the share $\llbracket q_{i',j} - \bar{q}_{i'} \rrbracket$ can be bound with the corresponding similarity share $\llbracket Sim(u_i, u_{i'}) \rrbracket$ to engage in the above shuffle protocol. By using the *Comparison* circuit to obtain the positions of Top-K similarity shares, RS and CSP can obtain the desired shares of $Sim(u_i, u_{i'})$ and $q_{i',j} - \bar{q}_{i'}$ directly.

The same idea can be applied to the computation of $P_s(q_{i,j})$ which requires the value of $q_{i,j'} - \bar{q}_{j'}$ for each service $s_{j'} \in SS(s_j)$. One notable difference here, however, is that after the computation of formula 3, the value $q_{i,j'} - \bar{q}_{j'}$ is only hold by CSP in the form of inner coding of garbled-circuit computation (see Fig. 6). Therefore, RS needs to generate a random number $r_{j'}$ and then cooperates with CSP to construct an *ADD* circuit to generate $q_{i,j'} - \bar{q}_{j'} + r_{j'}$ which is then given to CSP.

4.3. *QoS Prediction*

We are in the position to make the final QoS prediction which is also achieved by a composite garbled circuit whose general structure is depicted in Fig. 7. We first note that the Top-K

similar users (or services) returned by the secure Top-K query may actually contain some dissimilar users (or services) as their similarity values are negative. That is, the secure Top-K only returns $T(u_i)$ rather than $SU(u_i)$ for user u_i (see formula 5), but $SU(u_i)$ is required in formulae for QoS prediction. Instead of directly filtering these dissimilar users (or services), we introduce another garbled circuit to set negative similarity values to 0 so that formulae for QoS prediction can be correctly done on $T(u_i)$. As shown in Fig. 8, an ADD circuit is used to compute Sim_k which is additively secret shared, and then a MUX circuit is used to set negative Sim_k to 0. Given a MUX circuit which has two normal inputs, say x and y, and one special input called control bit b, its output is determined by the control bit b: if $b = 1$, then the output is x; otherwise, the output is y. In our circuit, the control bit is assigned with the sign of Sim_k which equals to 0 if $Sim_k < 0$ and 1 otherwise.

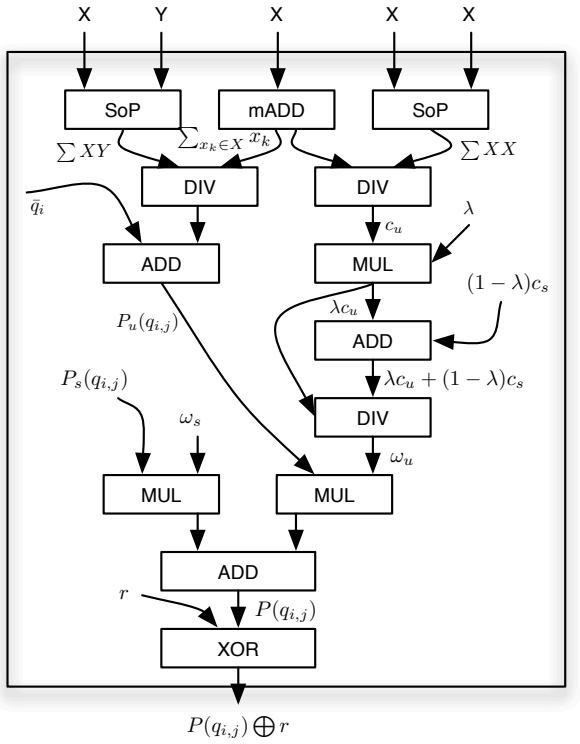

Fig. 7. Conceptual view of QoS prediction circuit

Note that the procedures of computing $\omega_u P_u(q_{i,j})$ and $\omega_s P_s(q_{i,j})$ are the same, so only the procedure of computing $\omega_u P_u(q_{i,j})$ is shown in Fig. 7 due to limited space. The input of this computation is two vectors: $X = [x_1, \cdots, x_{|SU(u_i)|}], Y = [y_1, \cdots, y_{|SU(u_i)|}]$ where $x_{i'} = Sim(u_i, u_{i'}), y_{i'} = q_{i',j} - \bar{q}_{i'}, u_{i'} \in SU(u_i)$. Note that both $x_{i'}$ and $y_{i'}$ are additively secret shared between RS and CSP, so they need to cooperatively construct an ADD circuit to generate $x_{i'}$ or $y_{i'}$. The circuit $mADD$ computes the sum of all elements in a vector. For formula 7, one addend $\bar{q}_i$ can be generated by an ADD circuit whose inputs are $-r_i$ hold by RS and $\bar{q}_i + r$ hold by CSP, while the other addend can be generated by the composition

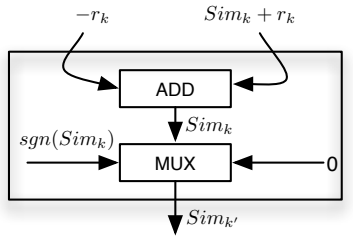

Fig. 8. Conceptual view of filtering circuit

of *SoP*, *mADD* and *DIV* circuits. For formula 8, the generation of the addend $\bar{q}_j$ is more complicated because RS does not hold $E(\bar{q}_j)$. To address this problem, CSP and RS holding $E(q_{i,j})$ first can cooperatively construct an *ADD* circuit to generate $g(q_{i,j})$. Then, they can construct a *SUB* circuit to generate $g(\bar{q}_j)$ as CSP holds $g(q_{i,j} - \bar{q}_j)$. The computation of $g(\omega_u)$ is straightforward and the corresponding procedure is shown in the right part of Fig. 7. One notable point is that RS knows λ, so this value can be directly given to a circuit.

The final prediction value $P(q_{i,j})$ is also a kind of private user information, so it should be kept secret to both RS and CSP. A simple method is to divide $P(q_{i,j})$ into two random shares. Instead of using additively secret sharing, *XOR* gate is used here as its overhead is nearly free. Specifically, RS generates and holds a random number r and CSP holds $P(q_{i,j}) \bigoplus r$ which is the output of the QoS prediction circuit. After that, RS and CSP send these shares to the user who can easily obtain the final QoS prediction value by executing exclusive-or operation on the shares she has just received.

5. Security Analysis

There are three kinds of participants in our framework: Users, RS and CSP. The messages received by each kind of participant during recommendation process are listed as follows.

- *Users:* The users who want to know the QoS of a particular service receive two random shares at the end of recommendation.
- *RS:* RS receives several encrypted data from the users before making prediction, some information during the running of Yao's garbled circuits, and some information during the secure Top-K query.
- *CSP:* CSP receives some information during the running of Yao's garbled circuits as well as the output of each circuit, and some information during the secure Top-K query.

To the users, the final prediction values are the final output of our framework, so they can efficiently simulate the random shares by generating numbers from the same domain. To RS and CSP, except for the process of executing Yao's garbled circuits and secure Top-K query, the only information revealed to them is either ciphertext or random shares. Clearly, this kind of information does not contain any privacy data, and we can simulate them by simply generating random numbers. In [27], the authors present a formal security proof for Yao's garbled circuits, which ensures the message received during the process of running Yao's

garbled circuits will not cause any information leakage. During the process of secure Top-K query, two participants received some comparison results. However, the comparisons occurs behind executing shuffling, so the comparison results do not contain any sensitive information.

6. Experiments

In this section, we present the complexity analysis and performance results of our privacy-preserving QoS prediction framework. We first note that a real Web services QoS dataset is available in [45], which includes QoS values of 5,825 real-world Web services observed by 339 users. This dataset is quite useful when studying the accuracy of QoS prediction. In this paper, however, our objective is to provide a privacy-preserving QoS prediction, so we focus on computational complexity rather than accuracy. In particular, the prediction accuracy of our framework is the same as that of non-privacy-preserving version presented in [46]. In terms of computational complexity, 339 users is too small to study the performance of our approach, so we create a synthetic dataset with $M = 10,000$ users and $N = 10,000$ services. Our framework is implemented in Java and tested on two computers (with Inter Xeon E5-2609 2.4GHz, 8GB RAM, Ubuntu Linux 12.04, and JRE 1.7) connected through a 100 Mbps LAN. The parameters used for the experiments are listed in Table 1. Note that, the runtime performance of our framework is affected by $|U|$ and $|S|$ (see formulae 1 and 3), which are actually changes as users provide new observed QoS values. In our experiments, however, we consider fixed $|U|$ and $|S|$ to make our results comparable. Specifically, we introduce two parameters α and β such that $|U| = \alpha M$ and $|S| = \beta N$.

6.1. *Complexity analysis*

Table 1. Parameter list used for the experiments

NP

Parameter	Symbol	Default Value
Number of users	M	5000
Number of services	N	5000
Common user ratio	α	0.02
Common service ratio	β	0.02
Bit size of QoS value	L	20
Top-K similar users or services	K	50
matrix density	ζ	10%
Message space of the cryptosystem	n	1024 bits
security parameter	r	80 bits

As the primary overheads of our framework comes from cryptographic operations and circuits execution, we give an analysis for each of them as follows.

Table. 2 summaries the computation and communication complexity of cryptographic operations required in our framework. To prepare the input for the garbled circuits of similarity calculation, RS needs to execute $O(M|S|+N|U|)$ times encryption on random numbers and executes $O(M|S| + N|U|)$ times multiplication and $O(N|U|)$ times exponentiation on ciphertext. After that, she sends these ciphertext to CSP, which costs $O(M|S| + N|U|)$ communication overheads. Meanwhile, CSP needs to receive these ciphertext from RS, so

the communication overheads is also $O(M|S| + N|U|)$. Besides, CSP needs to decrypt these ciphertext to obtain the shares of the input of garbled circuits for similarity computation, which costs $O(M|S| + N|U|)$ times decryption. In the secure Top-K query, both RS and CSP need $O(M + N)$ times encryption and decryption, and the data transfer between them requires $O(M+N)$ communication overheads. To the users, the main overheads is to encrypt their private data and transfer the ciphertext to RS. The computation and communication overheads are both $O(M + N)$. Note that, M and N have the same contribution to the computation and communication complexity, and this was also verified by our experiments, so we only report the experimental results for the effect of M on our framework in this paper due to limited space.

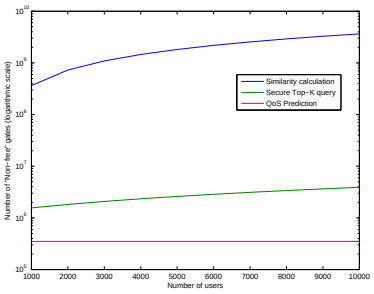

Fig. 9. Number of "Non-free" gates required in different steps of the proposed framework

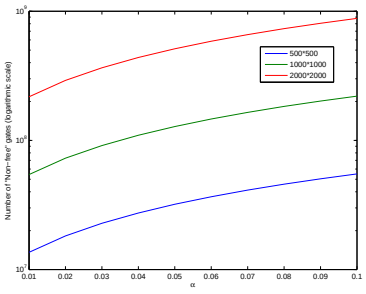

Fig. 10. α's effect on the number of "Non-free" gates required in the similarity calculation

The number of "Non-free" gate is an important metric to measure the overhead of Yao's garbled circuits [19]. Table. 2 summaries the number of "Non-free" gates required in different steps of our framework. The main overheads in the process of similarity computation is the execution of $O(M|S| + N|U|)$ MUL and ADD circuits, which requires $O((L^2 + L)(M|S| + N|U|))$ "Non-free" gates. In the secure Top-K query, $O(M + N)$ ADD and COM circuits are required, which costs $O((M + N)L)$ "Non-free" gates. The garbled circuits for QoS prediction requires $O(K)$ ADD, MUL and DIV circuits, so the corresponding "Non-free" gate is $O(K(L^2 + L))$. Fig. 9 depicts the number of "Non-free" gates required in different steps. Clearly, most "Non-free" gates are consumed by similarity calculation. Besides, the number of "Non-free" gates required by similarity calculation and secure Top-K query increases as the number of users increases, but QoS prediction only needs nearly constant number of "Non-free" gates, which coincides with the conclusion presented in Table 2. Since most "Non-free"

Table 2. Computational and communication complexity for RS, CSP and Users. E: Encryption, D: Decryption, Mul: Multiplication, Exp: Exponentiation, C: Communication

NP

	RS	CSP	Users
E	$O(M\|S\| + N\|U\|)$	$O(M + N)$	$O(MN\zeta)$
D	$O(M + N)$	$O(M\|S\| + N\|U\|)$	–
Mul	$O(M\|S\| + N\|U\|)$	$O(M + N)$	–
Exp	$O(N\|U\|)$	–	–
C	$O(M\|S\| + N\|U\|)$	$O(M\|S\| + N\|U\|)$	$O(MN\zeta)$

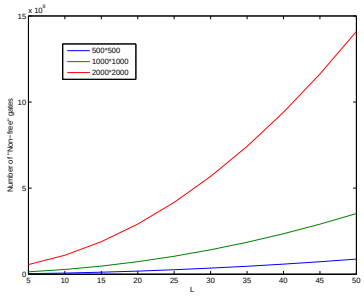

Fig. 11. L's effect on the number of "Non-free" gates required in the similarity calculation

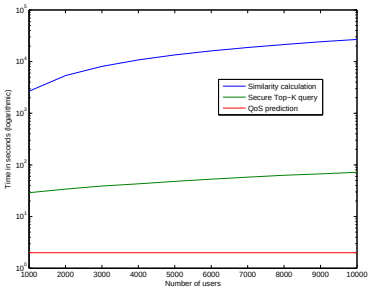

Fig. 12. Average runtime of different steps of the proposed framework

gates are spent on similarity calculation, we also conduct experiments to study the effect of other variable parameters on the number of "Non-free" gates required in similarity calculation. From Fig. 10, we can see that the number of "Non-free" gates increases almost linearly as α increases no matter how many users and services are taken into account (e.g., the red line represents the case where there are 2,000 users and 2,000 services). The results shown in Fig.11 are also in accord with our theoretical analysis.

6.2. *Runtime Performance*

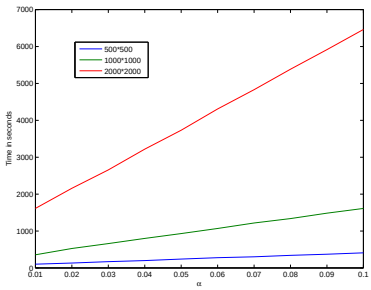

Fig. 13. α's effect on the average runtime of similarity calculation

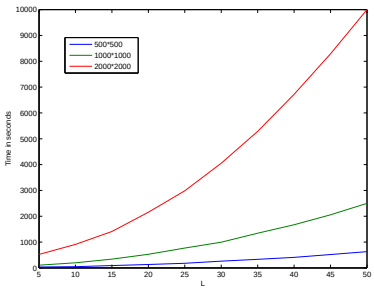

Fig. 14. L's effect on the average runtime of similarity calculation

Fig. 12 depicts the average runtime of different steps of our framework for predicting one QoS value over 10 runs. Again, we can see that most time are spent on similarity calculation,

Table 3. "Non-free" gates required in different steps of the proposed framework

NP

Step	"Non-free" gates
Step	"Non-free" gates
Similarity Calculation	$O((L^2+L)(M\|S\|+N\|U\|))$
Secure Top-K query	$O((M+N)L)$
QoS Prediction	$O(K(L^2+L))$

but the required time is still linear in the number of users in the recommendation system. Fig. 13 and Fig. 14 show α's and L's effects on the average runtime of similarity calculation, respectively. Clearly, the average runtime increases linearly as α increases, and quadratically as L increases.

Finally, we report the average overall runtime that our framework takes to generate a recommendation for a single user. When there are 1,000 users and 5,000 services, the overall runtime is about 45 minutes. When there are 10,000 users and 5,000 services, the overall runtime is about 7.5 hours. As mentioned earlier, this computation can be done offline, so a user can generally obtain a recommendation shortly after her request without any delays. In addition, the overall runtime can be also reduced significantly through code optimization, parallelization and dedicated hardware. In summary, the overall runtime of the proposed framework is linear with the number of users and the number of services, and thus possesses a good scalability.

7. Related Work

In [35], the authors present a standard user-based collaborative filtering algorithm to predict QoS. In [46], the authors combine user-based and item-based collaborative filtering, which greatly improve the accuracy of prediction. After that, various techniques have been designed and adapted to further improve the quality of prediction [7, 21, 38, 40, 41, 44, 47, 48]. The focus of these works is different from ours, but they can serve as the foundation of our framework.

The need for privacy protection for recommendation systems, particularly those using collaborative filtering techniques, triggered research efforts in the past years. Shokri *et al.* present a recommendation system built on distributed aggregation of user profiles, which suffers from the trade-off between privacy and accuracy [37]. In [31, 30], Nikolaenko *el al* consider two basic problems in the privacy-preserving recommendation: matrix factorization or ridge regression. For the first problem, they propose a solution based on Yao' garbled circuit. For the second problem, they design a hybrid method by combing YGC and additively homomorphic encryption. Their work can serve as the building blocks to design privacy-preserving recommendation systems. However, our work does not need matrix factorization and ridge regression. In [13], the authors present a solution to generate private recommendations via homomorphic encryption and data packing. Our framework differs from their work in the following ways: 1) in their work, the complex non-linear computations are conducted by users as they are assumed to know which items are rated by other users, however, this assumption is too strong as it can lead to information leakage. In our framework, these non-linear computations are done by RS and CSP through garbled circuits, which guarantees no information leakage; 2) in their work, a threshold is set to determine which users belongs to the Top-K similar user, which is also impractical. On the contrary, our secure Top-K query does not need such a threshold; 3) both user similarity and item similarity are considered in our framework while they only take user similarity into account, which makes our problem much more difficult than theirs.

On the other hand, there is also a number of works that aim at database privacy under the differential privacy paradigm [11, 12, 29]. This kind of work is orthogonal to ours, as we have different threat models. In differential privacy, a database owner has full access to the

data in the clear. The privacy threat arises from releasing a function over the data to a third party, which may use it to infer data values of users in the database. In our work, however, both RS and CSP pose a threat to user data.

8. Conclusion

To the best of our knowledge, this is the first piece of work that designs a privacy-preserving collaborative Web services QoS prediction framework: users send their encrypted observed QoS to the recommendation system who cooperates with a crypto service provider to generate a recommendation in the encrypted domain by running cryptographic protocols. The final recommendation and all the intermediate results during recommendation are either ciphertext or random shares, which ensures our framework is secure. To guarantee prediction quality, we have built our framework on the approach presented in [46]. As this approach requires a number of non-linear computations, we have employed Yao's garbled circuit instead of pure homomorphic encryption. To realize efficient QoS prediction, we have selected FasterGC as the implementation base and have extended it by adding several useful modules for garbled circuits. We have given a formal proof of the security of our framework, analyzed its computational and communication cost, and conducted extensive simulations. Both theoretical and empirical results show that our framework provides a secure and efficient collaborative Web services QoS prediction.

Acknowledgment This work was partially supported by Natural Science Foundation of China (Grant Nos. 61572336, 61303019, 61402313).

References

1. M. Alrifai, D. Skoutas, and T. Risse, "Selecting Skyline Services for QoS-Based Web Service Composition," Proc. Int'l Conf. World Wide Web (WWW'10), pp. 11-20, 2010.
2. M. Alrifai and T. Risse, "Combining global optimization with local selection for efficient QoS-aware service composition," Proc. Int'l Conf. World Wide Web (WWW'09), pp. 881-890, 2009.
3. D. Ardagna and B. Pernici, "Adaptive Service Composition in Flexible Processes," IEEE Trans. Software Engineering, vol. 33, no. 6, pp. 369-384, June 2007.
4. D. Beaver, Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. Journal of Cryptology, no 4, pp. 75-122. 1991.
5. M. Bellare, V. Hoang, P. Rogaway, "Foundations of garbled circuits," Proc. ACM conference on Computer and communications security (CCS'12), pp. 784-796, 2012
6. A. Ben-David, N. Nisan, B. Pinkas, "Fariplaymp: a system for secure multi-party computation," Proc. ACM conference on Computer and communications security (CCS'08), pp. 257-266, 2008
7. X. Chen, Z. Zheng, X. Liu, Z. Huang, and H. Sun, "Personalized QoS-Aware Web Service Recommendation and Visualization," IEEE Trans. Services Computing, vol. 6, no. 1, pp. 35-47, January-March 2013.
8. T. Cormen, C. Leiserson, R. Rivest and C.Stein, "Introduction to algorithms," the MIT press
9. J. W. Crenshaw, "Integer square roots," Embedded Systems Programming, Feb. 1998
10. I. Damgård, M. Geisler, and M. Krøgaard, "Efficient and secure comparison for on-line auctions," Proc. Australasian Conf. Information Security and Privacy (ACSIP'07), pp. 416-430, 2007
11. C. Dwork, "Differential privacy," Proc. Automata, Languages and Programming (ICALP'06), 2006
12. C. Dwork and J. Lei, "Differential privacy and robust statistics," Proc. ACM STOC Symposium on Theory of Computing (STOC'09), 2009
13. Z. Erkin, T. Veugen, T. Toft, and R.L. Lagendijk, "Generating Private Recommendations Effi-

ciently Using Homomorphic Encryption and Data Packing," IEEE Trans. Information Forensics and Security, vol. 7, no. 3, pp. 1053-1066, June 2012
14. S. Even, O. Goldreich, and A. Lempel, "A randomized protocol for signing contracts," Commun. ACM, vol. 28, no.6, 1985
15. S. Garg, S. Versteeg, R. Buyya, "A framework for ranking of cloud computing services," Future Generation Comp. Syst. (FGCS) vol. 29, no. 4, pp 1012-1023, 2013
16. O. Goldrich, Foundations of Cryptography: Volume 2, Basic Applications, Cambridge university press, 2004
17. D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, D. Savio, "Interacting with the SOA-Based Internet of Things: Discovery, Query, Selection, and On-Demand Provisioning of Web Services," IEEE T. Services Computing (TSC) vol. 3, no. 3, pp. 223-235, 2010
18. W. Henecka, A. Sadeghi, T. Schneider et al, "Tasty: tool for automating secure two-party computations," Proc. ACM conference on Computer and communications security (CCS'10), pp. 451-462, 2010
19. Y. Huang, D. Evans, J. Katz et al, "Faster secure two-party computation using garbled circuits," Proc. USENIX Security Symposium, 2011
20. Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," Proc. CRYPTO'03, 2003
21. Y. Jiang, J. Liu, M. Tang, and X. F. Liu. An effective web service recommendation method based on personalized collaborative filtering. Proc. IEEE Int'l Conf. Web Services (ICWS'11), pp. 211-218, 2011
22. D. Knuth, The Art of Computer Programming, Vol. 2, Semuninumerical Algorithms, 3rd ed. Addison-Wesley, 1997
23. V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free XOR gates and applications," Proc. Automata, Languages and Programming (ICALP'08). Springer, 2008
24. V. Kolesnikov, A.-R. Sadeghi, and T. Schneider, "Improved garbled circuit building blocks and applications to auctions and computing minima," Proc. Cryptology and Network Security (CANS'09), Springer, 2009
25. B. Kreuter, A. Shelat, C. Shen, "Billion-gate secure computation with malicious adversaries," Proc. USENIX conference on Security symposium, 2012
26. Y. Lindell, B. Pinkas, N. Smart, "Implementing two-party computation efficiently with security agains malicious adversaries." Proc. International conference on Security and Cryptography for Networks (SCN'08), pp. 2-20, 2008
27. Y. Lindell and B. Pinkas, "A proof of Yao's protocol for two-party computation," J. Cryptology, 2009
28. D. Malkhi, N. Nisan and B. Pinkas, "Fairplay-secure two-party computation systems," Proc. USENIX conference on Security symposium, pp. 287-302. 2004
29. F. McSherry and I. Mironov, "Differentially private recommender systems: Building privacy into the Netflix prize contenders," Proc. ACM SIGKDD international conference on Knowledge discovery in data mining (KDD'09), 2009
30. V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft. "Privacy-preserving ridge regression on hundreds of millions of records," Proc. IEEE Symposium on Security and Privacy (S&P'13), 2013
31. V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft. "Privacy-Preserving matrix factorization," Proc. ACM conference on Computer and communications security (CCS'13), 2013
32. P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," Proc. Advances in Cryptology (EUROCRYPT' 99), pp. 223-238, 1999
33. B. Pinkas, T. Schneider, N. Smart, "Secure two-party computation is practical," Proc. International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT'09), pp. 250-267, 2009
34. M. Rabin, "How to exchange secrets by oblivious trarnsfer," Technical Report TR-81, Aiken

Computation Laboratory, Harvard University, 1981
35. L. Shao, J. Zhang, Y. Wei, J. Zhao, B. Xie, and H. Mei, "Personalized qos prediction forweb services via collaborative filtering," Proc. IEEE Int'l Conf. Web Services (ICWS'07), pp. 439-446, 2007
36. Y. Shen, J. Zhu, X. Wang, L. Cai, X. Yang, B. Zhou, "Geographic Location-Based Network-Aware QoS Prediction for Service Composition," Proc. IEEE Int'l Conf. Web Services (ICWS'13), pp. 66-74, 2013
37. R. Shokri, P. Pedarsani, G. Theodorakopoulos, and J. Hubaux, "Preserving privacy in collaborative filtering through distributed aggregation of offline profiles," Proc. ACM conference on Recommender systems (RecSys'09), pp. 157-164, 2009
38. M. Tang, Y. Jiang, J. Liu, and X.Liu, "Location-Aware Collaborative Filtering for QoS-Based Service Recommendation", Proc. IEEE Int'l Conf. Web Services (ICWS'12), pp. 202-209, 2012
39. A.C.-C. Yao. How to generate and exchange secrets. Proc. IEEE Symposium on Foundations of Computer Scirence (FOCS'86), pp. 162-167, 1986
40. L. Yao, Q.Z. Sheng, A. Segev, J. Yu, "Recommending Web Services via Combining Collaborative Filtering with Content-Based Features," Proc. IEEE Int'l Conf. Web Services (ICWS'13), pp. 42-49, 2013
41. Q. Yu, Z. Zheng, and H. Wang, "Trace Norm Regularized Matrix Factorization for Service Recommendation," Proc. IEEE Int'l Conf. Web Services (ICWS'13), pp. 34-41, 2013
42. T. Yu, Y. Zhang, and K.-J. Lin, "Efficient algorithms for Web Services Selection with End-to-End QoS Constraints," ACM Trans. Web, vol. 1, no. 1, pp. 1-26, 2007
43. L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalaggnanam, and H. Chang, "QoS-Aware Middleware for Web Services Composition," IEEE Transactions on Software Engineering, vol. 30, no. 5, pp. 311-327, 2004
44. Q. Zhang, C. Ding, and C.-H. Chi. Collaborative filtering based service ranking using invocation histories. Proc. IEEE Int'l Conf. Web Services (ICWS'11), pp. 195-202, 2011
45. Y. Zhang, Z. Zheng, and M.R. Lyu, "Exploring Latent Features for Memory-Based QoS Prediction in Cloud Computing," Proc. IEEE Symposium on Reliable Distributed Systems (SRDS 2011), Madrid, Spain, Oct.4-7, 2011
46. Z. Zheng, H. Ma, M.R. Lyu, I. King, "WSRec: A Collaborative Filtering Based Web Service Recommender System," Proc. IEEE Int'l Conf. Web Services (ICWS'09), pp. 437-444, 2009
47. Z. Zheng, X. Wu, Y. Zhang, M.R. Lyu, and J. Wang, 'QoS Ranking Prediction for Cloud Services," IEEE Trans. Parallel and Distributed Systems, vol. 24, no. 6, pp. 1213-1222, June 2013
48. Z. Zheng, H. Ma, M.R. Lyu, and I. King, "Collaborative Web Service QoS Prediction via Neighborhood Integrated Matrix Factorization," IEEE Trans. Services Computing, vol. 6, no. 3, pp. 289-299, July-September 2013