Computers and Electrical Engineering 59 (2017) 153–164

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

Achieving reliable and secure services in cloud computing environments☆

Qin Liu[a], Guojun Wang[b,*], Xuhui Liu[a], Tao Peng[c], Jie Wu[d]

[a] College of Computer Science and Electronic Engineering, Hunan University, Changsha, Hunan Province, 410082, PR China
[b] School of Computer Science and Educational Software, Guangzhou University, Guangzhou, Guangdong Province, 510006, PR China
[c] School of Information Science and Engineering, Central South University, Changsha, Hunan Province, 410083, P R China
[d] Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122, USA

ARTICLE INFO

Article history:
Received 21 April 2016
Revised 7 October 2016
Accepted 7 October 2016
Available online 19 October 2016

Keywords:
Cloud computing
Reliability
Security
Comprehensive transmission model
Two-phase resource sharing protocol

ABSTRACT

In cloud computing environments, resources stored on the cloud servers are transmitted in the form of data flow to the clients via networks. Due to the real-time and ubiquitous requirements of cloud computing services, how to design a sophisticated transmission model to ensure service reliability and security becomes a key problem. In this paper, we first propose a Comprehensive Transmission (CT) model, by combining the Client/Server (C/S) mode and the Peer-to-Peer (P2P) mode for reliable data transmission. Then, we design a Two-Phase Resource Sharing (TPRS) protocol, which mainly consists of a pre-filtering phase and a verification phase, to *efficiently* and *privately* achieve *authorized resource sharing* in the CT model. Extensive experiments have been conducted on the synthetic data set to verify the feasibility of our protocol.

1. Introduction

Cloud computing as a promising computing paradigm, allows users to enjoy user-controlled services transparently and seamlessly. In cloud computing, everything is a service (XaaS). That is, computer resources (e.g., hardware, software, and data) are delivered as services that can be subscribed and unsubscribed by customers over the Internet in a pay-as-you-go fashion. In the NIST cloud definition framework [1], cloud computing offers three kinds of service models, i.e., Software as a Service (SaaS) that allows the cloud customers to control only application configurations, Platform as a Service (PaaS) that allows the cloud customers to control the hosting environments, and Infrastructure as a Service (IaaS) that allows the cloud customers to control everything except the hardware infrastructure.

Fig. 1 shows a typical cloud computing environment, where all of the resources stored on the cloud servers will be transmitted in the form of data flow to the clients. The users can access any desired resources on demand, anytime and anywhere, using various kinds of devices connected to the Internet. Cloud computing, as an evolved paradigm of distributed computing, parallel computing, grid computing, and utility computing, has a lot of merits such as fast deployment, pay-for-use, high availability, high scalability, rapid elasticity, low costs, and so on. However, its unique features bring new challenges to service reliability [2,3].

☆ Reviews processed and recommended for publication to the Editor-in-Chief by Area Editor Dr. G. Martinez.
* Corresponding author.
E-mail address: csgjwang@gmail.com (G. Wang).

http://dx.doi.org/10.1016/j.compeleceng.2016.10.005
0045-7906/© 2016 Elsevier Ltd. All rights reserved.

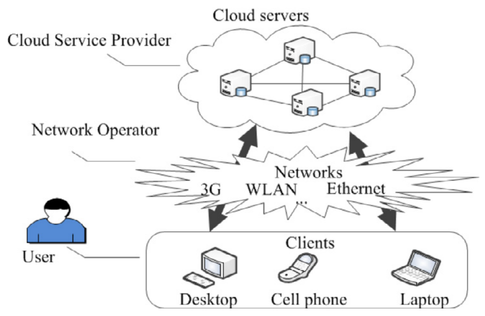


Fig. 1. A typical cloud environment. The clients held by the user are network-connected devices such as cell phones and laptops; the cloud servers maintained by the Cloud Service Provider (CSP) are regular personal computers; and the networks managed by the Network Operator support various kinds of network access methods, e.g., WLAN, 4G, etc.

First of all, cloud computing requires high real-time services. In cloud computing environments, resources are dynamically scheduled over networks in a block-streaming way. The delay or lack of a single block during data transmission will cause the failure of the service delivery. Secondly, cloud computing requires ubiquitous services. The users are promised to enjoy services at anytime and at any place using any type of device. In certain circumstances, the time delay for data transmission between the cloud servers and the clients may be intolerant. Such examples include the cloud servers becoming overloaded or the clients having low-speed Internet access. To ensure the quality of the service, an alternative solution is to establish a peer-to-peer connection between nearby clients for direct data transferring.

In this paper, we propose a Comprehensive Transmission (CT) model to ensure service reliability in cloud environments. The CT model, which combines the Client/Server (C/S) mode and the Peer-to-Peer (P2P) mode for data transmission, allows the clients to access resources from the cloud servers in the C/S mode or other clients in the P2P mode. The CT model makes full use of the advantages of both modes: On one hand, the centralized resource management in C/S mode is in favor of access control and system security; On the other hand, P2P resource sharing can reduce the workload on the cloud servers, which mitigates the risk of a single point of failure [4–6].

However, different users have different access rights for various resources in cloud computing environments. Although the C/S mode can easily achieve fine-grained access control under the cloud servers' centralized control, *efficiently* and *privately* achieving *authorized resource sharing* in the P2P mode becomes a problem. For efficiency, we should ensure that a client can quickly find appropriate resources. For privacy, we should protect the privacy of a client while searching resources (i.e., which resources it is authorized to access). For authorized resource sharing, we should ensure that a client can access resources only after obtaining authorization from the cloud servers.

To this end, we design a Two-Phase Resource Sharing (TPRS) protocol, which mainly consists of a pre-filtering phase and a verification phase. In the TPRS protocol, the clients are further classified as *requesters* and *providers* for certain resources. The pre-filtering phase applies the Secure Dot-Product (SDP) protocol [7] to let a requester *efficiently* and *privately* find *candidate* providers. Then, the verification phase applies the Fuzzy Vault (FV) [8] and Attribute-Based Encryption (ABE) [9] techniques to let each party *privately* verify the authenticity of the results in the pre-filtering phase for authorized resource sharing. The main contributions of our work are as follows:

- To achieve reliable and secure services in cloud computing environments, we propose a CT model by combining the C/S mode and the P2P mode for data transmission.
- We are among the first to consider the problem of efficient, private, and authorized resource sharing in cloud computing environments. We design a TPRS protocol, which consists of a pre-filtering phase and a verification phase.
- Extensive experiments have been conducted on the synthetic data set to verify the feasibility and effectiveness of the proposed protocol.

Paper organization. We provide our models and definitions in Section 2 before introducing technique preliminaries in Section 3. We present the pre-filtering phase in Section 4 before illustrating the verification phase in Section 5. Then, we analyze the security of the proposed protocol in Section 6. After providing an evaluation in Section 7, we introduce related works in Section 8. Finally, we conclude this paper in Section 9.

2. Models and definitions

2.1. Comprehensive transmission model

The CT model mainly consists of two kinds of entities: the cloud servers and the clients, as shown in Fig. 2. In the CT model, we mainly consider wireless networks, where many Access Points (APs) are deployed for data transmission. We assume that the clients close to an AP form a *group*.

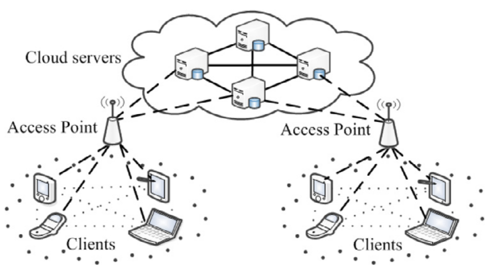


Fig. 2. The CT model.

In cloud computing, services are delivered over networks in a multi-tenancy fashion, where application programs and data resources are shared across a number of tenants. Therefore, it is convenient for a client to request resources from nearby clients. As a trade-off between service reliability and security, the CT model allows a client to cache resources for a certain period of time, t, other than *one-time-use*. During t, the client can be viewed as an authorized *provider* to provide resources to other clients (referred as to *requesters*) in a group.

In the CT model, resource sharing is based on time, thus we split time into time periods of length t, with t_i denoting the i-th time period. Each time period is further divided into N time slices, with $T_{i,j}$ denoting the jth time slice of t_i for $1 \leq j \leq N$. Suppose that the set of universal resources in the system is $\mathcal{R} = \{a_1, a_2, \ldots, a_d\}$. Each client, u, identified by a unique identity, ID_u, is associated with a *resource coordinate*, $RS_u = \{a_{u1}, a_{u2}, \ldots, a_{um}\}$, where $RS_u \subseteq \mathcal{R}$ denotes a set of resources, which u has the right to access. Let $|\cdot|$ represent the cardinality of a set. Therefore, we have $|\mathcal{R}| = d$ and $|RS_u| = m$.

To access the desired resources, client u first sends its resource coordinate RS_u to the cloud server, which will determine whether u is authorized to access resources in RS_u or not. If so, the cloud server will provide resources to u directly or authorize u to request resources from other clients. For the former case, the cloud server will generate a set of *time-based proofs*, $\{prf_a^{t_i}\}_{a\in RS_u}$ at the end of transmission. Here, $prf_a^{t_i}$ denotes u is authorized to provide resource a to other clients during time period t_i. For the latter case, the cloud server first determines the time slice $T_{i,j}$, during which u is authorized to access resources from other clients. Then, it generates a set of *time-based tokens*, $\{(\mathcal{TK}_{u,a}^{T_{i,j}}, \mathcal{TK}_{u,a}^{T_{i,j+1}})\}_{a\in RS_u}$, and a set of *time-based decryption keys* $\{dk_{u,a}^{T_{i,j}}\}_{a\in RS_u}$ for u. Here, the time-based tokens are used to construct a time-based fuzzy vault $\mathcal{FV}_{\mu}^{T_{i,j}}$ for authenticating the providers' identity, and the time-based decryption keys are used to recover message M for confirming the authenticity of RS_μ to the providers. After obtaining authorization from the cloud server, u runs the TPRS protocol to achieve authorized resource sharing in an efficient and private way.

2.2. Threat model

In the threat model, the cloud server is a trusted entity responsible for generating system parameters and keys. The clients can be divided into requesters and providers, which may be trusted or untrusted. For the cloud server, we should ensure authorized resource sharing. That is, requester μ possesses time-based token, $\{(\mathcal{TK}_{\mu,a}^{T_{i,j}}, \mathcal{TK}_{\mu,a}^{T_{i,j+1}})\}$, and time-based decryption key, $dk_{\mu,a}^{T_{i,j}}$, only after μ is authorized by the cloud server to access resource a at time slice $T_{i,j}$. In brief, requester μ cannot forge its resource coordinate RS_μ to access unauthorized resources at any time.

For client u, associated with RS_u, its *collaborator* and *close collaborator* can be defined as follows:

Collaborator and close collaborator. Client v, associated with RS_v, is u's collaborator if $|RS_u \cap RS_v| > 0$; v is u's close collaborator if $|RS_u \cap RS_v| \geq \theta$, where $\theta \in [1, m]$ is a threshold value predetermined by u, and $RS_u \cap RS_v$ denote a set of common resources in both RS_u and RS_v.

For trusted clients, we need to protect the privacy of their resource coordinates from those that are not their collaborators. That is, we should ensure that the intersection $RS_u \cap RS_v$ is disclosed to u and v only when they are collaborators. Furthermore, we should ensure that u will share resources with v only when they are *close collaborators*.

We assume that the communication channels are secured under existing security protocols, e.g., SSL and SSH, during information transferral. Our protocol is considered to fail if any of the following cases is true:

Case 1. The client u can forge its resource coordinate RS_u by colluding with other untrusted clients to access unauthorized resources at time slice $T_{i,j}$.

Case 2. The client u's resource coordinate RS_u is disclosed to those which are not u's collaborators.

Case 3. The client u accesses resources from those which are not u's close collaborators.

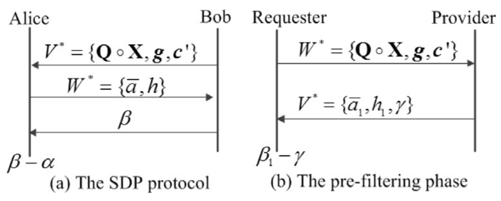

Alice
Bob
$V^* = \{\mathbf{Q} \circ \mathbf{X}, \boldsymbol{g}, \boldsymbol{c}'\}$
$W^* = \{\bar{a}, h\}$
β
$\beta - \alpha$
(a) The SDP protocol
Requester
Provider
$W^* = \{\mathbf{Q} \circ \mathbf{X}, \boldsymbol{g}, \boldsymbol{c}'\}$
$V^* = \{\bar{a}_1, h_1, \gamma\}$
$\beta_1 - \gamma$
(b) The pre-filtering phase

Fig. 3. The SDP protocol and the pre-filtering protocol.

3. Technique preliminaries

3.1. Secure dot-product protocol

The SDP protocol proposed by Ioannidis et al. [7] allows two parties to compute the dot-product of two vectors without directly exposing the vectors to each other. Suppose that Alice has a d-dimensional vector $\mathbf{w}$ and Bob has a d-dimensional vector $\mathbf{v}$, and only Alice is interested of the result of the dot product of $\mathbf{v}$ and $\mathbf{w}$, denoted as $\mathbf{w} \circ \mathbf{v}$.

The working process of the SDP protocol is shown in Fig. 3(a): 1) Bob generates a fuzzy matrix $\mathbf{X}$ by adding some random noises to $\mathbf{v}$, and then generates a random matrix $\mathbf{Q}$ and some random noises R, based on which he calculates $\mathbf{Q} \cdot \mathbf{X}$ and auxiliary information b, $\mathbf{g}$ and $\mathbf{c}'$. The random noise R and the auxiliary information b will be kept secret. The message from Bob to Alice is $V^* = \{\mathbf{Q} \cdot \mathbf{X}, \mathbf{g}, \mathbf{c}'\}$. 2) Alice generates a fuzzy vector $\mathbf{w}'$ by adding a random noise α to $\mathbf{w}$, and then computes the intermediate results, $\bar{a}$ and h, based on V^* and $\mathbf{w}'$. The random noise α will be kept secret. The message from Alice to Bob is $W^* = \{\bar{a}, h\}$. 3) Bob calculates and sends β to Alice, based on W^*, auxiliary information b, and random noises R. 4) Alice calculates $\beta - \alpha$ to obtain $\mathbf{w} \circ \mathbf{v}$. By using linear algebraic technique, the SDP protocol provides a powerful tool for secure dot-product calculation, and incurs a low overhead [10].

3.2. Fuzzy vault

FV was proposed by Juels et al. [8], where Alice places a secret value κ in a fuzzy vault and *locks* it using a set A of elements, so that Bob can *unlock* the vault to obtain κ using a set B of elements only when A and B overlap substantially. The hardness is based on the polynomial reconstruction problem, a special case of the Reed-Solomon List Decoding problem [11]. Specifically, to lock κ under set A, Alice selects a polynomial p in a single variable x, such that p encodes κ in some way. Then, she computes evaluations of p on the elements of A, by treating the elements of A as distinct x-coordinate values, to form $R_A = \{x_i, y_i\}_{x_i \in A \wedge y_i = p(x_i)}$, and creates a number of random noises that do not lie on p to form $R_N = \{X_i, Y_i\}_{X_i \notin A \wedge Y_i \neq p(X_i)}$. Alice will send $R = R_A \cup R_N$ to Bob. If B overlaps substantially with A, Bob can identify many points in R that lie on p, thus reconstructing p exactly and recovering κ.

It is worth noticing that FV is vulnerable to the cross-matching attacks [12]. If the attacker has access to multiple vaults locked by the same set A he can easily identify the randomness. Cryptography solutions can address the cross-matching attack issues in the fuzzy vault, and yet they will involve the issue of unlocking key storage and management. In the field of bio-cryptography, PinSketch with a biometrics-key binding mechanism can address the cross-matching problems without dependence on the unlocking key. Interested readers are referred to following references [13–15] for the latest developments on this topic.

3.3. Attribute-based encryption

In ABE [16,17], users are identified by a set of *attributes*, and a ciphertext, associated with an *attribute-based access structure*, can be decrypted by the users whose attributes satisfy the access structure. For example, given an access structure $\mathbb{A} = (a_1 \wedge a_2) \vee a_3$, either users with attributes a_1 and a_2, or users with attribute a_3 can decrypt the ciphertext.

In cloud computing, the resources associated with each user are equal to user attributes in ABE. In the verification phase, the provider will utilize the hierarchical ABE scheme proposed in [9] (referred to as HABE) to authenticate the requester's resource coordinate. The HABE scheme, constructed based on the bilinear map, is proven to be semantically secure under the random oracle model and the Bilinear Diffie-Hellman (BDH) assumption [18]. The access structure in HABE is expressed as a disjunctive normal form (DNF). For example, access structure $\mathbb{A} = \overset{N}{\underset{i=1}{\vee}} (CC_i) = \overset{N}{\underset{i=1}{\vee}} (\overset{n_i}{\underset{j=1}{\wedge}} a_{ij})$ consists of N conjunctive clauses, $CC_1, \ldots, CC_N$, where the ith conjunctive clause CC_i is expressed as $a_{i1} \wedge a_{i2} \wedge \ldots \wedge a_{in_i}$. The users that possess all of the attributes in CC_i can decrypt the ciphertext. The main merit of the HABE scheme is that it requires only a constant number of bilinear map operations for encrypting a message, and the length of the ciphertext is related to the number of conjunctive clauses instead of the number of attributes in the access structure.

4. The pre-filtering phase

In this section, we will illustrate the construction of the pre-filtering phase, which is based on the SDP protocol [7,10]. Given a requester μ, associated with RS_μ, and a provider ν, associated with RS_ν, μ will know whether $|RS_\mu \cap RS_\nu| \geq \theta$ or not at the end of the pre-filtering phase, without exposing their resource coordinates to each other.

Suppose that the universal resources $\mathcal{R} = \{a_1, \ldots, a_d\}$, and that requester μ holds a vector $\mathbf{w}$ and provider ν holds $\mathbf{v}$, both in d dimensions. Let w_i and v_i denote the ith element in $\mathbf{w}$ and $\mathbf{v}$, respectively. w_i (v_i) will be set to 1 if resource $a_i \in RS_\mu$ ($a_i \in RS_\nu$); otherwise, it is set to 0. Given a security parameter $S \geq 2$ that controls the amount of randomness to hide resource coordinates, the pre-filtering phase works as follows (see Fig. 3(b)):

1) Requester μ constructs a random $S \times S$ matrix $\mathbf{Q}$, and constructs a $(d+1)$-dimensional vector, $\mathbf{w}'$, where $w'_i = w_i$ for $1 \leq i \leq d$ and $w'_{d+1} = 1$. Then, μ generates a $S \times (d+1)$ matrix $\mathbf{X}$. Let $\mathbf{x}_i$ denotes the ith row of $\mathbf{X}$. We have:
$$\mathbf{x}_i = \begin{cases} \mathbf{w}', & i = r_0, \\ \mathbf{k}_i, & \forall i \neq r_0, \end{cases}$$
where $r_0 \in [1, S]$ is a random number, and $\mathbf{k}_1, \ldots, \mathbf{k}_{r_0-1}, \mathbf{k}_{r_0+1}, \mathbf{k}_S$ are $S-1$ random $(d+1)$-dimensional vectors. Then compute:
$$\mathbf{Q} \cdot \mathbf{X}, \quad \mathbf{c}' = \mathbf{c} + \mathbf{f} \cdot r_1 \cdot r_2$$
$$\mathbf{g} = \mathbf{f} \cdot r_1 \cdot r_3, \quad b = \sum_{i=1}^{S} \mathbf{Q}_{ir_0}$$
where $\mathbf{c} = \sum_{i=1, i \neq r_0}^{S} (\mathbf{x}_i \cdot \sum_{j=1}^{S} \mathbf{Q}_{ji})$, $\mathbf{f}$ is a random $(d+1)$dimensional vector, and r_1, r_2, r_3 are random numbers. The auxiliary information b and random noises $\{r_2, r_3\}$ will be kept for calculating β_1 in Step 3). The message from μ to all its nearby clients is denoted as $W^* = \{\mathbf{Q} \cdot \mathbf{X}, \mathbf{c}', \mathbf{g}\}$.
2) Provider ν constructs a $(d+1)$-dimensional vector $\mathbf{v}'$ where $v'_i = v_i$ for $1 \leq i \leq d$ and $v'_{d+1} = \alpha$ with α a random value, and then chooses a random number $\rho > 0$ to compute $\rho \cdot \mathbf{v}'$. Based on the scaled vector, ν calculates:
$$\mathbf{y} = \mathbf{Q} \cdot \mathbf{X} \cdot (\rho \cdot \mathbf{v}'), \qquad z = \sum_{i=1}^{S} y_i$$
$$\bar{a}_1 = z - \mathbf{c}' \circ (\rho \cdot \mathbf{v}'), \qquad h_1 = \mathbf{g} \circ (\rho \cdot \mathbf{v}')$$
where y_i denotes the ith element of vector $\mathbf{y}$. Then, ν calculates $\gamma = \alpha + \rho \cdot \theta + \sigma$, where $0 \leq \sigma < \rho$ is a random number and θ is the threshold value. The message from ν to μ is $V^* = \{\bar{a}_1, h_1, \gamma\}$.
3) Requester μ computes: $\beta_1 = \frac{\bar{a}_1 + h_1 \cdot R}{b}$, where $R = r_2 / r_3$. If $\beta_1 - \gamma > 0$, ν will be treated as a *candidate provider*, and will enter the verification phase.

Proof of correctness. Since $\gamma = \alpha + \rho \cdot \theta + \sigma$, we have $\beta_1 - \gamma = \beta_1 - \alpha - \rho \cdot \theta - \sigma$. Let T denote the dot product of $\mathbf{w}$ and $\mathbf{v}$. Provider ν scales the vector $\mathbf{v}'$ with a random number ρ, thus $\beta_1 - \alpha = \rho \cdot T$ and $\beta_1 - \gamma = \rho \cdot T - \rho \cdot \theta - \sigma = \rho(T - \theta) - \sigma$. Since $\rho > \sigma$, $\beta_1 - \gamma \geq \theta$ implies $T > \theta$. ■

Security sketch. For the provider, the information they obtained from the requester is no more than that in the SDP protocol [7], and thus our protocol can preserve the same level of privacy. In addition, the requester will know $\gamma = \alpha + \rho \cdot \theta + \sigma$. As proven in [10], since γ is hidden by two random numbers ρ and σ, the probability of revealing α is essentially low. ■

5. The verification phase

For quick reference, we provide a summary of the most relevant notations in Table 1. Based on the FV technique [8] and the HABE scheme [9], we construct the verification protocol as follows:

1) *Setup*$(K) \to (PK, MK, s)$: The cloud server takes a security parameter K as the input, and outputs a system public key PK, a system master key MK, and a root secret key $s \in \mathbb{Z}_q^*$ as follows:
$$PK = (q, \mathbb{G}_1, \mathbb{G}_2, Q_0, \hat{e}, P_0, P_1, H_1, H_2, H_3)$$
$$MK = (k_0, k_1, K_1)$$
where $(q, \mathbb{G}_1, \mathbb{G}_2, \hat{e})$ are the outputs of a BDH parameter generator $\mathcal{IG}$ [18], P_0 is a random generator of $\mathbb{G}_1$, P_1 is a random element in $\mathbb{G}_1$; H_1: $\{0,1\}^* \to \mathbb{G}_1$ and H_2: $\mathbb{G}_2 \to \{0,1\}^L$ for some L are random oracles; $H_3 : \mathbb{G}_1 \to \mathbb{Z}_q^*$ is a hash function; k_0 and k_1 are random elements in $\mathbb{Z}_q^*$; $Q_0 = k_0 P_0 \in \mathbb{G}_1$, and $K_1 = k_0 P_1 \in \mathbb{G}_1$. PK is publicly available, but MK and s are kept secret.

Table 1
Summary of notations.

Notation	Description
$u, v, w,$	Clients
$\mu, \nu,$	Resource requester, resource provider
$\mathcal{R}, d$	Universal resources, size of $\mathcal{R}$
RS_u, m	u's resource coordinate, size of RS_u
$t_i, T_{i,j}$	Time period i, the jth time slice in t_i
$\mathcal{TK}_{u,a}^{T_{i,j}}$	u's token for attribute a at $T_{i,j}$
$dk_{u,a}^{T_{i,j}}$	u's time-based decryption key
$prf_a^{t_i}$	Proof of possessing resource a at t_i
$\mathcal{FV}_u^{T_{i,j}}$	Time-based fuzzy vault

2) $GenToken(PK, s, ID_\mu, a, T_{i,j}, T_{i,j+1}) \to (\mathcal{TK}_{\mu,a}^{T_{i,j}}, \mathcal{TK}_{\mu,a}^{T_{i,j+1}})$: The cloud server takes system public key PK, the root secret key s, μ's identity ID_μ, and time slice $T_{i,j}$ as inputs, and generates a time-based token $\mathcal{TK}_{\mu,a}^{T_{i,j}} = H_3(s_a^{T_{i,j}} H_1(ID_\mu)) \in \mathbb{Z}_q^*$. Here, $s_a^{T_{i,j}}$ can be calculated in a hierarchical way:

$$s_a = H_s(a), s_a^{t_i} = H_{s_a}(t_i), s_a^{T_{i,j}} = H_{s_a^{t_i}}(T_{i,j}) \tag{1}$$

where attribute a, time period t_i, and time slice $T_{i,j}$ will be expressed as a bit string, and $H_s : \{0,1\}^* \to \mathbb{Z}_q^*$, $H_{s_a} : \{0,1\}^* \to \mathbb{Z}_q^*$, and $H_{s_a^{t_i}} : \{0,1\}^* \to \mathbb{Z}_q^*$ are keyed hashes with s, s_a, and $s_a^{t_i}$ as their secret keys, respectively. In the same way, the cloud server generates a time-based token $\mathcal{TK}_{\mu,a}^{T_{i,j+1}}$ for μ.

3) $GenProof(PK, s, a, t_i) \to (prf_a^{t_i})$: The cloud server takes system public key PK, root secret key s, and a time period t_i as inputs, and generates $s_a^{t_i} \in \mathbb{Z}_q^*$ with Eq. (1). Then, it sets the time-based proof $prf_a^{t_i} = s_a^{t_i}$ for provider ν.

4) $GenKey(PK, MK, ID_\mu, a, T_{i,j}) \to (dk_{\mu,a}^{T_{i,j}})$: The cloud server takes system public key PK, system master key MK, requester μ's identity ID_μ, resource a, and time slice $T_{i,j}$ as inputs, and generates a decryption key $dk_{\mu,a}^{T_{i,j}} = (K_\mu, K_{\mu,a}^{T_{i,j}})$ as follows:

$$K_\mu = k_1 k_\mu P_0, \qquad K_{\mu,a}^{T_{i,j}} = K_1 + k_1 k_\mu H_1(a||T_{i,j})$$

where ID_μ is expressed as a bit string, $k_\mu \in \mathbb{Z}_q^*$ is a random element associated with ID_μ, and $H_1(a||T_{i,j}) \in \mathbb{G}_1$ denotes resource a's public key at time slice $T_{i,j}$. The notation "||" denotes concatenation. For example, the result of "0011"||"1100" is "00111100".

5) $Lock(\kappa, p, \{\mathcal{TK}_{\mu,a}^{T_{i,j}}, \mathcal{TK}_{\mu,a}^{T_{i,j+1}}\}_{a \in RS_\mu}) \to (\mathcal{FV}_\mu^{T_{i,j}})$: To lock secret $\kappa \in \mathbb{Z}_q^*$, requester μ picks a random polynomial p of $(\theta - 1)$ degree and sets $p(0) = \kappa$. Then, it calculates $p(\mathcal{TK}_{\mu,a}^{T_{i,j+1}})$ for each resource $a \in RS_\mu$, and generates the time-based fuzzy vault $\mathcal{FV}_\mu^{T_{i,j}} = \{(\mathcal{TK}_{\mu,a}^{T_{i,j}}, p(\mathcal{TK}_{\mu,a}^{T_{i,j+1}}))\}_{a \in RS_\mu}$.

6) $Unlock(PK, \mathcal{FV}_\mu^{T_{i,j}}, \{prf_a^{t_i}\}_{a \in RS_\nu}) \to (\kappa)$: For each time-based proof $prf_a^{t_i}$, provider ν first calculates $s_a^{T_{i,j}} = H_{s_a^{t_i}}(T_{i,j}) = H_{prf_a^{t_i}}(T_{i,j})$ and then tests whether $H_3(s_a^{T_{i,j}} H_1(ID_\mu))$ appears in $\mathcal{FV}_\mu^{T_{i,j}}$ or not. If so, it calculates $\mathcal{TK}_{\mu,a}^{T_{i,j+1}} = H_3(s_a^{T_{i,j+1}} H_1(ID_\mu))$, where $s_a^{T_{i,j+1}} = H_{s_a^{t_i}}(T_{i,j+1}) = H_{prf_a^{t_i}}(T_{i,j+1})$. If $|RS_\mu \cap RS_\nu| \geq \theta$, it can obtain sufficient points on polynomial p, and use $\{(\mathcal{TK}_{\mu,a}^{T_{i,j+1}}, p(\mathcal{TK}_{\mu,a}^{T_{i,j+1}}))\}_{a \in RS_\mu \cap RS_\nu}$, to reconstruct p and get $p(0)$ as confirmation message κ.

7) $Encrypt(PK, \mathbb{A}, M, T_{i,j}) \to (C_{\mathbb{A}}^{T_{i,j}})$: As in the HABE scheme [9], the access structure $\mathbb{A}$ is expressed in the DNF form. Specifically, to verify whether requester μ is authorized to access all of the resources in $RS_\mu = \{a_{\mu 1}, \ldots, a_{\mu m}\}$ at time slice $T_{i,j}$, the provider ν will specify an access structure $\mathbb{A} = a_{\mu 1} \wedge \ldots \wedge a_{\mu m} \wedge T_{i,j}$. Then, ν encrypts a message $M \in \{0, 1\}^L$ with access structure $\mathbb{A}$ by calculating Eq. 2:

$$U_0 = rP_0, \tag{2a}$$

$$U_1 = r \sum_{a \in RS_\mu} H_1(a||T_{i,j}), \tag{2b}$$

$$V = M \oplus H_2(\hat{e}(Q_0, rmP_1)) \tag{2c}$$

where $r \in \mathbb{Z}_q^*$ is a random number, and $H_1(a||T_{i,j}) \in \mathbb{G}_1$ denotes resource a's public key at time slice $T_{i,j}$. The ciphertext $C_{\mathbb{A}}^{T_{i,j}}$ consists of (U_0, U_1, V).

8) $Decrypt(PK, C_{\mathbb{A}}^{T_{i,j}}, \{dk_{\mu,a}^{T_{i,j}}\}_{a \preceq \mathbb{A}}) \rightarrow (M)$: Given ciphertext $C_{\mathbb{A}}^{T_{i,j}}$, requester μ uses its decryption keys $\{dk_{\mu,a}^{T_{i,j}}\}_{a \preceq \mathbb{A}}$ to recover data M with Eq. (3):

$$M = V \oplus H_2\left(\frac{\hat{e}(U_0, \sum_{a\in RS_\mu} K_{\mu,a}^{T_{i,j}})}{\hat{e}(K_\mu, U_1)}\right) \tag{3}$$

Note that $K_\mu = k_1 k_\mu P_0$ and $K_{\mu,a}^{T_{i,j}} = K_1 + k_1 k_\mu H_1(a||T_{i,j})$. Thus, the correctness of Eq. (3) can be proven as follows:

$$\begin{aligned} H_2\left(\frac{\hat{e}(U_0, \sum_{a\in RS_\mu} K_{\mu,a}^{T_{i,j}})}{\hat{e}(K_\mu, U_1)}\right) &= H_2\left(\frac{\hat{e}(rP_0, \sum_{a\in RS_\mu}(K_1 + k_1 k_\mu H_1(a||T_{i,j})))}{\hat{e}(k_1 k_\mu P_0, r\sum_{a\in RS_\mu} H_1(a||T_{i,j}))}\right) \\ &= H_2\left(\frac{\hat{e}(rP_0, mK_1)\hat{e}(rP_0, \sum_{a\in RS_\mu} k_1 k_\mu H_1(a||T_{i,j}))}{\hat{e}(k_1 k_\mu P_0, r\sum_{a\in RS_\mu} H_1(a||T_{i,j}))}\right) \\ &= H_2(\hat{e}(rP_0, mK_1)) \end{aligned}$$

Thus, we have:

$$\begin{aligned} V \oplus H_2\left(\frac{\hat{e}(U_0, \sum_{a\in RS_\mu} K_{\mu,a}^{T_{i,j}})}{\hat{e}(K_\mu, U_1)}\right) &= M \oplus H_2(\hat{e}(Q_0, rmP_1)) \oplus H_2(\hat{e}(rP_0, mK_1)) \\ &= M \oplus H_2(\hat{e}(Q_0, rmP_1)) \oplus H_2(\hat{e}(Q_0, rmP_1)) = M \end{aligned}$$

6. Security analysis

As defined in Section 2, our TPRS protocol is considered to fail if any of the following cases are true:

Case 1. The client u has the ability to forge its resource coordinate RS_u to access unauthorized resources at time slice $T_{i,j}$.

In our TPRS protocol, the provider will provide resources to the requester only after the requester can decrypt and send back M. Therefore, we consider that Case 1 will not happen if the following propositions hold:

Proposition 1. *The keys produced by the GenKey algorithm are secure.*

Proposition 2. *The ciphertext produced by the Encrypt algorithm is semantically secure.*

For Proposition 1, we prove that our *GenKey* algorithm is as secure as the *GenKey* algorithm in the HABE scheme (referred to as the *HGenKey* algorithm). If requester μ is authorized to access resource a at time slice $T_{i,j}$, it will obtain a time-based decryption key $dk_{\mu,a}^{T_{i,j}} = \{K_\mu, K_{\mu,a}^{T_{i,j}}\}$. Firstly, the way to generate K_μ is the same in both algorithms. Secondly, given the system public key PK, the system master key MK, μ's identity ID_μ, and the concatenation of attribute a and time slice $T_{i,j}$, $a||T_{i,j}$, as inputs of the *HGenKey* algorithm, the produced key $K_{\mu,a}^{T_{i,j}}$ is the same as that of our *GenKey* algorithm. As proven in [9], due to the BDH assumption, malicious users cannot obtain MK, even if all of them collude. Thus, Proposition 1 is correct.

For Proposition 2, recall that message M is encrypted to $V = M \oplus H_2(\hat{e}(Q_0, rmP_1))$. Therefore, an adversary needs to construct $\hat{e}(Q_0, rmP_1) = \hat{e}(U_0, K_1)^m$ to recover M. From the *GenKey* algorithm, we know that the only occurrence of K_1 is in $K_{\mu,a}^{T_{i,j}}$. Then, we consider the case that malicious clients work independently, or collude to compromise data security.

The *Encrypt* algorithm is considered to be insecure if adversary $\mathcal{A}$, whose resource coordinates do not satisfy the access structure $\mathbb{A}$ at time slice $T_{i,j}$, can recover M. We have the following assumptions for ease of presentation: Adversary $\mathcal{A}$ has requested keys on all but one of the resources $a_{u1}, \ldots, a_{u(k-1)}, a_{u(k+1)}, \ldots, a_{um}$ in $\mathbb{A}$ for client u, and has requested a key on the missing resources a_{vk} for client v, at time slice $T_{i,j}$. The only occurrence of K_1 is in $K_{\mu,a}^{T_{i,j}}$, so the adversary has to use keys requested for client u and v for bilinear map, yielding for some ξ:

$$\hat{e}\left(U_0, \sum_{l=1,l\neq k}^{m} K_{u,a_{ul}}^{T_{i,j}} + K_{v,a_{vk}}^{T_{i,j}} + \xi\right) = \hat{e}(U_0, mK_1)\hat{e}(rP_0, \xi)\hat{e}(K_v, rH_1(a_{vk}||T_{i,j}))\hat{e}(K_u, r\sum_{l=1,l\neq k}^{m} H_1(a_{ul}||T_{i,j}))$$

where $H_1(a||T_{i,j})$ denotes a's public key at time slice $T_{i,j}$. To obtain $\hat{e}(U_0, K_1)^m$, the last three elements have to be eliminated. Note that K_v and K_u are known to adversary $\mathcal{A}$, but r is randomly chosen by the encrypter for the ciphertext $C_{\mathbb{A}}^{T_{i,j}}$. The adversary cannot know $rH_1(a_{vk}||T_{i,j})$ or $r\sum_{l=1,l\neq k}^{m} H_1(a_{ul}||T_{i,j})$, even if it knows $U_1 = r\sum_{a\in RS_\mu} H_1(a||T_{i,j})$ due to the BDH assumption. Therefore, adversary $\mathcal{A}$ cannot recover M from V. Thus, Proposition 2 is correct, and Case 1 will not happen.

Case 2. The client u's resource coordinate RS_u is disclosed to client v which is not u's collaborator.

The pre-filtering phase is based on the SDP protocol, the security of which has been proven in [7]. Thus, the entities involved only know whether $|RS_u \cap RS_v| \geq \theta$ or not, without exposing their resource coordinates. In the verification phase, the message from requester μ to provider ν is a time-based fuzzy vault $\mathcal{FV}_\mu^{T_{i,j}} = \{(\mathcal{TK}_{\mu,a}^{T_{i,j}}, p(\mathcal{TK}_{\mu,a}^{T_{i,j+1}}))\}_{a\in RS_\mu}$. If ν and μ are not collaborators at time slice $T_{i,j}$, we know that $RS_\mu \cap RS_\nu = 0$. For each proof $prf_{a_{\nu k}}^{t_i}$, ν cannot obtain $s_{a_{\nu k}}^{t_i}$ satisfying $a_{\nu k} \in$

Table 2
Parameter setting.

Parameter	Description	Value
d	Number of universal resources	{50, 100}
n	Number of clients	$40 \sim 200$
m	Size of the resource coordinate RS_u	$5 \sim 20$
S	Security parameter	{2, 5}
θ	Threshold value	$\{m/2, m\}$
L	The length of message M	64B

Table 3
The computation cost at the cloud server.

	$m = 5$	$m = 10$	$m = 15$	$m = 20$
GenToken	259 ms	423 ms	598 ms	758 ms
GenKey	304 ms	584 ms	863 ms	1149 ms
GenProof	0.49 ms	0.71 ms	0.93 ms	1.20 ms

RS_{μ}. Due to the security of the hash functions, it cannot obtain $s_{a_{\mu l}}^{T_{i,j}}$ for any resource $a_{\mu l} \in RS_{\mu}$. Due to the BDH assumption, it cannot generate token $\mathcal{TK}_{a_{\mu l}}^{T_{i,j}}$ for any resource $a_{\mu l} \in RS_{\mu}$, and thus cannot know which resources μ is requesting for.

The message from provider ν to requester μ is a ciphertext $C_{\mathbb{A}}^{T_{i,j}}$ encrypted with the *Encrypt* algorithm, which is proven to be semantically secure in Case 1. Therefore, μ can recover M to know ν's resource coordinate only when it possesses sufficient decryption keys, which implies that ν is a collaborator of μ. Thus, Case 2 will not happen.

Case 3. The client u accesses resources from those which are not u's close collaborators.

In our protocol, requester μ requests resources from provider ν only when ν unlocks and sends back κ. As the proven in Case 2, ν can calculate $\mathcal{TK}_{a_{\nu k}}^{T_{i,j+1}}$ satisfying $\mathcal{TK}_{a_{\nu k}}^{T_{i,j+1}} = \mathcal{TK}_{a_{ul}}^{T_{i,j+1}}$ only when $a_{\nu l} = a_{\mu k}$. That is, ν cannot forge proofs for the unauthorized resources. Due to the hardness of the Reed–Solomon List Decoding problem [11], ν has the ability to recover p only when it obtains sufficient points lying on p. Thus, Case 3 will not happen.

7. Evaluation

In this section, we will evaluate the performance of the TPRS protocol in terms of computation cost. The cloud server is running on a Linux system with Intel Xeon X3430 at 2.4GHz CPU and 4.0 GB RAM, and the client is running on a Linux system with Intel Core i3-2310M at 2.10 GHz CPU and 4.0 GB RAM. The cryptographic algorithms in the verification phase are implemented with JPBC library [19]. We use a 160-bit elliptic curve $y^2 = x^3 + x$ over a 512-bit finite field, in which q is a 160-bit length prime, and the length of element in $\mathbb{G}_1$ is 512-bit. The parameters used in the experiments are shown in Table 2.

7.1. The costs at the cloud server

In our experiments, the *Setup* algorithm takes about *330 ms* for performing a constant number of exponentiation operations. However, the *Setup* algorithm is run only during system initialization, and hence can be done once and for all. Then, the cloud server runs algorithms *GenToken* and *GenKey* to generate tokens and decryption keys for a requester, or runs the *GenProof* algorithm to generate proofs for a provider. The average computation costs for these algorithms are shown in Table 3. From Table 3, we know that most of processing time is spent in the *GenKey* algorithm, where the most expensive operation is the calculation of $H_1(a||T_{i,\,j})$. As an improvement, we allow the cloud server to cache the result of the random oracle, so that $H_1(a||T_{i,\,j})$ is only executed once regardless of how many clients are requesting a at time slice $T_{i,\,j}$. Therefore, the computation time of the *GenKey* algorithm does not grow linearly as the number of clients, n, increases.

To verify the scalability of the proposed protocol, we will test the performance of cloud server while n ranges from 40 to 200. We consider random distribution and power-law distribution for the resources in RS_u. From Figs. 4,5 and6, we know that parameter m has the greatest impact on the computation time at the cloud server. Furthermore, our protocol has better performance in pow-law distribution compared with random distribution. However, the distribution has less influence on the *GenToken* algorithm. The reason is, for two clients sharing the same resource a at time slice $T_{i,\,j}$, only the calculation for $s_a^{T_{i,j}}$ can be shared in the *GenToken* algorithm.

7.2. The costs at the requester

In our TPRS protocol, requester μ, associated with RS_{μ}, first participates in the pre-filter phase to find out candidate providers. From Table 4, we know that the computation cost increases as either d or S increases. In the worst case, it takes less than 1ms to finish this step.

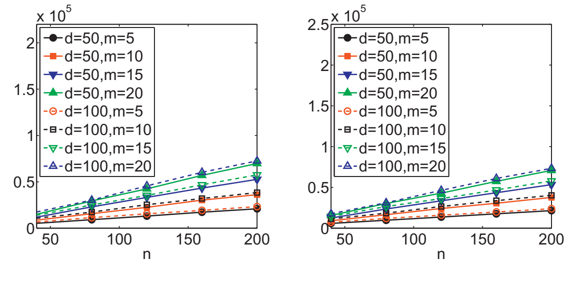


Fig. 4. Computation time for the *GenKey* algorithm (ms).

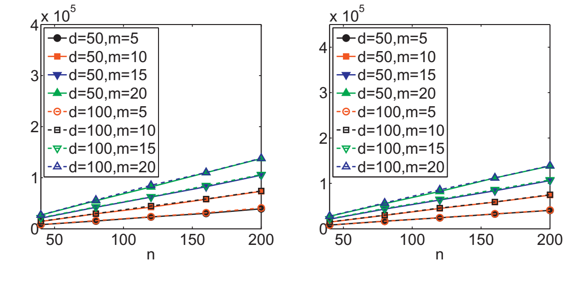


Fig. 5. Computation time for the *GenToken* algorithm (ms).

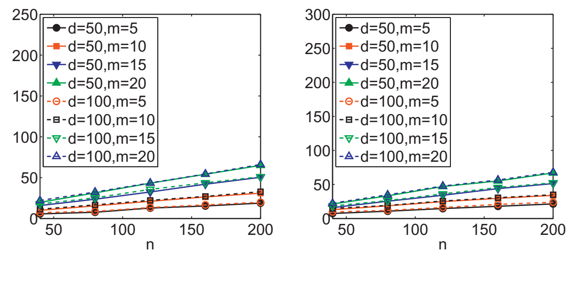


Fig. 6. Computation time for the *GenProof* algorithm (ms).

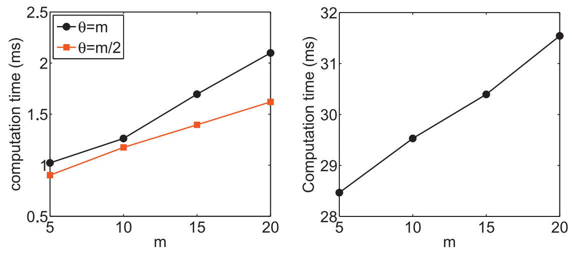
Table 4
The computation cost in the pre-filtering phase.

	$S=2$		$S=5$	
	$d=50$	$d=100$	$d=50$	$d=100$
Requester	0.173 ms	0.367 ms	0.232 ms	0.429 ms
Provider	0.064 ms	0.137 ms	0.097 ms	0.169 ms

The computation costs for algorithms *Lock* and *Decrypt* are shown in Fig. 7, from which we know that the computation time of the *Lock* algorithm mainly depends on parameters m and θ, and the computation cost of the *Decrypt* algorithm increases as m increases. However, both algorithms are very efficient.

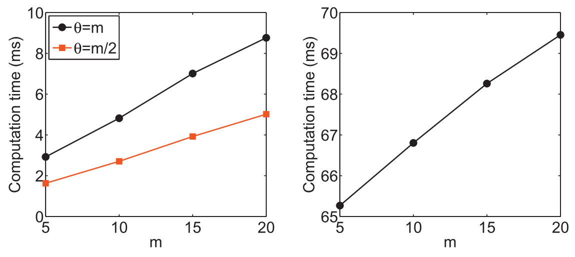
7.3. The costs at the provider

In the verification phase, provider ν first runs the *Unlock* algorithm to recover confirmation message κ. Then, it runs the *Encrypt* algorithm to generate a ciphertext. The most expensive operations in the above algorithms include the computation of a random oracle in $\mathbb{G}_1$, as well as the computation of exponentiation. As an improvement, we allow the provider to request this information from the cloud server rather than calculating them independently. The computation costs for



(a) The *Lock* algorithm. (b) The *Decrypt* algorithm.

Fig. 7. Computation cost in the verification phase at the requester.



(a) The *Unlock* algorithm. (b) The *Encrypt* algorithm.

Fig. 8. Computation cost at the provider in the verification phase.

algorithms *Unlock* and *Encrypt* are shown in Fig. 8, from which we know that the computation costs in both algorithms will grow as m increases. Furthermore, the performance of the *Unlock* algorithm is also affected by the threshold value θ.

Summary. The pre-filtering phase costs less than 1 ms in total. Thus, the requester can quickly screen a majority of providers whose resource coordinates have less than θ intersection within its own resource coordinate. After obtaining authorization from the cloud server, the total consumed time for the provider is within 100 ms, and the total consumed time for the requester to authenticate one provider is within 40 ms in the worst case. Therefore, we can set the length of each time slice to 200 ms. For a time period consisting of 100 time slices, the time duration for caching data is about 20 s.

8. Related work

The term *cloud computing* was first coined by Google CEO Eric Schmidt in 2006, and was immediately popular within the industry. Although cloud computing has overwhelming superiorities over traditional computing models, the adoption of clouds is still far from expected. The main reason is that customers worry that their data may be leaked or tampered by the cloud vendors. Therefore, improving the service reliability and security in cloud computing environments has always been the focus. For example, for reliable cloud services, Zhao et al. [20] introduced a low latency fault tolerance middleware to provide fault tolerance for distributed applications deployed within clouds. Jhawar et al. [21] proposed a comprehensive approach to transparently deliver fault-tolerance applications running on virtual machine instances in clouds. Liu et al. [22] presented a consistency as a service model, where the users were allowed to verify whether the CSP provided the promised level of consistency or not. However, existing work mainly focuses on the C/S transmission mode, where the clients request all of the resources from the cloud servers. The centralized resource management in C/S mode is in favour of access control and system security, but is more likely to cause a single point of failure. This paper proposed a CT model, which allows a client to access resources from either the cloud server in the C/S mode, or other clients in the P2P mode.

Based on the CT model, our research problem focuses on how to efficiently and privately achieve authorized resource sharing in the P2P mode. The problem of privately finding resources is similar to the secure friend discovery problem in social networks [10,23–25]. For example, Dong et al. [10] proposed a privacy-preserving and verifiable secure dot-product protocol based on the SDP protocol [7] and Homomorphic Encryption. With their protocol, a user's social profile will be protected during friend discovery in mobile social networks. Li et al. [23] proposed a privacy-preserving personal profile matching scheme for mobile social networks by utilizing the efficient private matching and Set Intersection technique and the Shamir secret sharing scheme. The main difference between our work and theirs is that we need to verify the authenticity of the resource coordinate to achieve authorized resource sharing. Therefore, we generate tokens and keys to the clients during authorization and apply the FV and ABE techniques for mutual authentication.

9. Conclusion and future work

Our CT model allows a client to cache resources for a certain period of time to achieve comprehensive resource sharing, which will better suit the ubiquitous and unconscious requirements of cloud services. In the CT model, our TPRS protocol allows the requesters to quickly screen out a majority of clients that do not meet their requirements in the pre-filter phase, and allows the requester and the provider to mutually verify the authenticity of each other's resource coordinate in the verification phase. In the empirical study, the interaction between clients takes about 200 ms. Therefore, our protocol enables the efficient, private, and authorized resource sharing in cloud computing.

However, the TPRS protocol treats all the cloud servers equally, and lets them share the system master key *MK* and the root secret key *s*. This makes it difficult to revoke a cloud server from the system. If a cloud server is compromised, *s* and *MK* will be exposed to the attacker. For system security, we need to revoke the compromised cloud server and run the *Setup* algorithm to reinitialize the whole system. Afterwards, we need to run the *GenToken, GenKey*, and *GenProof* algorithms for generating new tokens, keys, and proofs for all clients. This revocation process will introduce a heavy workload on the system. The preferred solution is to let each cloud server *i* associate personalized MK_i and s_i. Once cloud server *j* is revoked, we replace it with a new cloud server *i* and then utilize MK_i and s_i to re-generate keys to only the clients requesting resources from the revoked cloud server. Therefore, as part of our future work, we will try to incorporate the idea of distributed ABE to our protocol to achieve efficient revocation of the cloud servers.

Acknowledgments

This work was supported in part by NSFC grants 61632009, 61472451, 61402161, 61472131, 61272151, 61272546; Hunan Provincial Natural Science Foundation of China (Grant No. 2015JJ3046); by CERNET Innovation Project (Grant No. NGII20150408) and NSF grants CNS 1449860, CNS 1461932, CNS 1460971, CNS 1439672, CNS 1301774, and ECCS 1231461.

References

[1] Mell P, Grance T. The NIST definition of cloud computing. Commun ACM 2010;53(6):50.
[2] Liu Q, Tan CC, Wu J, Wang G. Towards differential query services in cost-efficient clouds. IEEE Trans Parallel Distrib Syst 2014;25(6):1648–58.
[3] Xia Z, Wang X, Sun X, Wang Q. A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data. IEEE Trans Parallel Distrib Syst 2015;27(2):340–52.
[4] Cesare S, Xiang Y, Zhou W. Control flow-based malware variantdetection. IEEE Trans Dependable Secure Comput 2014;11(4):307–17.
[5] Xiao M, Wu J, Huang L. Time-sensitive utility-based single-copy routing in low-duty-cycle wireless sensor networks. IEEE Trans Parallel Distrib Syst 2015;26(5):1452–65.
[6] Xie K, Wang L, Wang X, Xie G, Wen J, Zhang G. Accurate recovery of internet traffic data: a tensor completion approach. In: Proc. of the 35th annual IEEE international conference on computer communications (INFOCOM); 2016. p. 1–9.
[7] Ioannidis I, Grama A, Atallah MJ. A secure protocol for computing dot-products in clustered and distributed environments. In: Proc. of the international conference on parallel processing (ICPP; 2002. p. 379–84.
[8] Juels A, Sudan M. A fuzzy vault scheme. Des Codes Cryptograph 2006;38(2):237–57.
[9] Wang G, Liu Q, Wu J. Hierarchical attribute-based encryption for fine-grained access control in cloud storage services. In: Proc. of the ACM conference on computer and communications security (CCS); 2010. p. 735–7.
[10] Dong W, Dave V, Qiu L, Zhang Y. Secure friend discovery in mobile social networks. In: Proc. of the IEEE international conference on computer communications (INFOCOM); 2011. p. 1647–55.
[11] Guruswami V, Sudan M. Improved decoding of reed-solomon and algebraic-geometric codes. In: Proc. of the annual symposium on IEEE foundations of computer science (FOCS; 1998. p. 28–37.
[12] Scheirer WJ, Boult TE. Cracking fuzzy vaults and biometric encryption. In: Proc. of IEEE biometrics symposium; 2007. p. 1–6.
[13] Yang W, Hu J, Wang S. A delaunay quadrangle-based fingerprint authentication system with template protection using topology code for local registration and security enhancement. IEEE Trans Inf Forensics Secur 2014;9(7):1179–92.
[14] Li C, Hu J, Pieprzyk J, Susilo W. A new biocryptosystem-oriented security analysis framework and implementation of multibiometric cryptosystems based on decision level fusion. IEEE Trans Inf Forensics Secur 2015;10(6):1193–206.
[15] Li C, Hu J. A security-enhanced alignment-free fuzzy vault-based fingerprint cryptosystem using pair-polar minutiae structures. IEEE Trans Inf Forensics Secur 2016;11(3):543–55.
[16] Bethencourt J, Sahai A, Waters B. Ciphertext-policy attribute-based encryption. In: Proc. of the IEEE symposium on security and privacy (S&P); 2007. p. 321–34.
[17] Müller S, Katzenbeisser S, Eckert C. Distributed attribute-based encryption. In: Proc. of the annual international conference on information security and cryptology (ICISC); 2009. p. 20–36.
[18] Boneh D, Franklin M. Identity-based encryption from the weil pairing. In: Advances in cryptology CRYPTO; 2001. p. 213–29.
[19] Caro AD, Iovino V. JPBC: java pairing based cryptography. In: Proc. of the IEEE symposium on computers and communications (ISCC); 2011. p. 850–5.
[20] Zhao W, Melliar-Smith PM, Moser LE. Fault tolerance middleware for cloud computing. In: Proc. of international conference on cloud computing (CLOUD); 2010. p. 67–74.
[21] Jhawar R, Piuri V, Santambrogio M. Fault tolerance management in cloud computing: a system-level perspective. IEEE Syst J 2013;7(2):288–97.
[22] Liu Q, Wang G, Wu J. Consistency as a service: auditing cloud consistency. IEEE Trans Netw Serv Manage 2014;11(1):25–35.
[23] Li M, Cao N, Yu S, Lou W. Findu: privacy-preserving personal profile matching in mobile social networks. In: Proc. of the IEEE international conference on computer communications (INFOCOM); 2011. p. 2435–43.
[24] Zhang R, Zhang Y, Sun J, Yan G. Fine-grained private matching for proximity-based mobile social networking. In: Proc. of the IEEE international conference on computer communications (INFOCOM); 2012. p. 1969–77.
[25] Zhang L, Li X, Liu Y. Message in a sealed bottle: privacy preserving friending in social networks. In: Proc. of the international conference on distributed computing systems (ICDCS); 2013. p. 327–36.

Qin Liu received her B.Sc. in Computer Science in 2004 from Hunan Normal University, China, received her M.Sc. in Computer Science in 2007, and received her Ph.D. in Computer Science in 2012 from Central South University, China. She is an Assistant Professor in the College of Computer Science and Electronic Engineering at Hunan University, China.

Guojun Wang received his B.Sc. in Geophysics in 1992, M.Sc. in Computer Science in 1996, and Ph.D. in Computer Science in 2002, from Central South University, China. He is the Pearl River Scholar Professor of Guangdong Province at School of Computer Science and Educational Software, Guangzhou University, China. He is a distinguished member of CCF.

Xuhui Liu received his B.Sc. in Computer Science in 2012 from Taishan University, China. He is a postgraduate student at the College of Computer Science and Electronic Engineering, Hunan University, China. His research interests include security issues in cloud computing.

Tao Peng received her B.Sc. in Computer Science in 2004 from Xiangtan University, China, and received her M.Sc. in Circuits and Systems in 2007 from Hunan Normal University, China. She is a Ph.D. candidate at School of Information Science and Engineering, Central South University, China. Her research interests include network and information security issues.

Jie Wu is the Chair and a Laura H. Carnell Professor in the Department of Computer and Information Sciences at Temple University, Philadelphia, PA, USA. He was an IEEE ComputerSociety Distinguished Visitor, ACM Distinguished Speaker, and chair for the IEEE Technical Committee on Distributed Processing (TCDP). He is a CCF Distinguished Speaker and a Fellow of the IEEE.