An Efficient Privacy Preserving Keyword Search Scheme in Cloud Computing

Qin Liu[†], Guojun Wang[†‡*], and Jie Wu[‡]

[†]School of Information Science and Engineering
Central South University
Changsha 410083, Hunan Province, P. R. China
[*] Correspondence to: csgjwang@mail.csu.edu.cn

[‡]Department of Computer Science and Engineering
Florida Atlantic University
Boca Raton, FL 33431, USA

***Abstract*—A user stores his personal files in a cloud, and retrieves them wherever and whenever he wants. For the sake of protecting the user data privacy and the user queries privacy, a user should store his personal files in an encrypted form in a cloud, and then sends queries in the form of encrypted keywords. However, a simple encryption scheme may not work well when a user wants to retrieve only files containing certain keywords using a thin client. First, the user needs to encrypt and decrypt files frequently, which depletes too much CPU capability and memory power of the client. Second, the service provider couldn't determine which files contain keywords specified by a user if the encryption is not searchable. Therefore, it can only return back all the encrypted files. A thin client generally has limited bandwidth, CPU and memory, and this may not be a feasible solution under the circumstances. In this paper, we investigate the characteristics of cloud computing and propose an efficient privacy preserving keyword search scheme in cloud computing. It allows a service provider to participate in partial decipherment to reduce a client's computational overhead, and enables the service provider to search the keywords on encrypted files to protect the user data privacy and the user queries privacy efficiently. By proof, our scheme is semantically secure.**

***Keywords:* cloud computing; privacy preserving; searchable encryption; partial decipherment**

I. Introduction

Cloud computing [1], which dynamically provides reliable services over the Internet, is one of the 2009 Top 10 Strategic Technologies [2]. Recently, many academic and industrial organizations have started investigating and developing technologies and infrastructure for cloud computing. The representative cloud platforms include Amazon Elastic Compute Cloud (EC2) [3], Google App Engine [4], and Microsoft Live Mesh [5].

A user stores his personal files in a cloud, and retrieves them wherever and whenever he wants. We consider the following application: a user U pays a service provider S for a storage service in order to store his email messages, and later he wants to retrieve only emails containing certain keywords when he is traveling with a thin client, such as a wireless PDA and a mobile phone. It is trivial to do so when the email messages are stored in the form of a plain-text, which will introduce undesirable security and privacy risks. For example, U is a technician in Company A who takes in charge of after-sale services. He stores all the emails sent from the customers in a cloud when he is in his office with a desktop, and retrieves them to tackle the customer's service requests when he is out with a PDA. An attacker who intercepts and captures the communications is able to know the customer's privacy information as well as some important business secrets. What is worse is that an untrustworthy service provider is able to easily obtain all the information and sell it to the biggest rival of Company A. As described in Haclgiimfi et al [6], there are two main attacks under such a circumstance, i.e., outer attacks initiated by unauthorized outsiders and inner attacks initiated by untrustworthy service providers. In some cases, we couldn't fully trust a service provider, but still need its service. Therefore, it needs to provide some mechanisms to protect the user data privacy and the user queries privacy in cloud environment.

The simplest solution is to encrypt the emails before storing them in a cloud and send queries in the form of encrypted keywords. For example, a user may use his public key to encrypt the email message body and its keywords before sending it to a service provider, and then sends queries in the form of encrypted keywords to retrieve the email. Since the secret key is only known to the user himself, an attacker has no idea of the encrypted files and the user queries patterns. However, such a simple encryption scheme may introduce other problems: (1) It depletes too much CPU capability and memory power of the client during the encryption and decryption; (2) The service provider couldn't determine which emails contain keywords specified by a user if the encryption is not searchable. Therefore, it can only return back all the encrypted emails. Generally speaking, a thin client has only limited bandwidth, CPU and memory, therefore a simple encryption scheme couldn't work well under these circumstances.

In this paper, we dedicate to solve the above problems and propose an efficient privacy preserving keyword search scheme in cloud computing. Our contributions are threefold:

1) It supports keyword search on encrypted data. It enables a service provider to determine whether a given email contains certain keywords specified by a user, but have no idea of any information about both the specified keywords and the encrypted emails. It is able to protect the user data privacy and the user queries privacy efficiently

during the search process.

2) It is efficient and practical. It enables a service provider to participate in partial decipherment so as to reduce a user's computational overhead, but has no ability to recover the plain-text. It is well suited to be applied to cloud computing.
3) It is a provably secure scheme. We first define an efficient privacy preserving keyword search scheme, and give its security definition in the sense of semantic security. And then we construct the scheme based on bilinear maps, and prove that it is semantically secure assuming the BDH problem is hard [8].

This paper is structured as follows. We first review some important background and definitions in Section 2. We then describe an efficient privacy preserving keyword search scheme and give its security definition in Section 3. We construct the scheme based on bilinear maps and prove that it is semantically secure in Section 4. Finally we present our future work and conclude this paper in Section 5.

II. Preliminaries

A. Related Work

Boneh et al [7] introduce a public key encryption with keyword search (**PEKS**) scheme, which supports the keyword search on encrypted data. The application context is as follows: (1) Bob sends to Alice an email encrypted under Alice's public key; (2) Alice's email gateway wants to test whether the email contains the keyword *urgent* so that it could route the email accordingly; (3) But Alice does not want the email gateway to be able to decrypt her messages. Boneh et al define and construct a mechanism that enables a gateway to test whether the word *urgent* is a keyword in the email using a trapdoor provided by Alice, but learn nothing about the email. This notion could be applicable to cloud environment with some improvements.

In cloud environment, Alice and Bob might be the same person who pays for a storage service. If a user wants to retrieve emails containing certain keywords when he is traveling with a PDA, **PEKS** couldn't work well. According to **PEKS**, a service provider returns back encrypted emails after finishing the search, and a user needs to decrypt them by himself. Frequent decryption will deplete too much CPU capability and memory power of the client and lose critical virtue of cloud computing. For the sake of reducing a user's computational overhead, we allow a service provider to participate in partial decipherment, but keep the plain-text to be blind to the service provider. Under normal circumstances, a user stores encrypted files once, but retrieves and decrypts them for many times. Our approach could reduce a user's computational overhead largely.

Diament et al [11] first introduce the notion of an efficient dual receiver cryptosystem, which enables a ciphertext to be decrypted by two independent receivers. The main disadvantage of the dual receiver cryptosystem is that the server needs to send an auxiliary private key to a client for decrypting a partial ciphertext, which is insecure in the real environment. Recently, there is much work on keyword search on encrypted files, such as Song et al [9], Bennett et al [10], and Chang et al [12]. In this paper, we borrow the idea of partial decipherment, and propose an efficient privacy preserving keyword search scheme by improving **PEKS**, which requires no private key transmission and is more applicable to a cloud environment.

B. Related Definitions

In this section, we introduce some definitions in Boneh et al [8] to form the basis of our scheme. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two cyclic groups of some large prime order q. We view $\mathbb{G}_1$ as an additive group and $\mathbb{G}_2$ as a multiplicative group.

Definition 2.1 (Bilinear Maps): We call e a bilinear map if $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ is a map with the following properties:

1) Computable: There is a polynomial time algorithm to compute $e(g, h) \in \mathbb{G}_2$, for any $g, h \in \mathbb{G}_1$.
2) Bilinear: $e(g^x, h^y) = e(g, h)^{xy}$ for all $g, h \in \mathbb{G}_1$ and all $x, y \in \mathbb{Z}_q^*$.
3) Non-degenerate: if g is a generator of $\mathbb{G}_1$, then $e(g, g)$ is a generator of $\mathbb{G}_2$.

Definition 2.2 (BDH Parameter Generator): We say that a randomized algorithm $\mathcal{IG}$ is a BDH parameter generator if $\mathcal{IG}$ takes a sufficiently large security parameter $K > 0$, runs in polynomial time in K, and outputs the description of two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of the same prime order q and the description of a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$.

Definition 2.3 (BDH Problem): Given a random element $g \in \mathbb{G}_1$, as well as g^x, g^y, and g^z, for some $x, y, z \in \mathbb{Z}_q^*$, compute $e(g, g)^{xyz} \in \mathbb{G}_2$.

Definition 2.4 (BDH Assumption): If $\mathcal{IG}$ is a BDH parameter generator, the advantage $Adv_{\mathcal{IG}}(\mathcal{B})$ that an algorithm $\mathcal{B}$ has in solving the BDH problem is defined to be the probability that $\mathcal{B}$ outputs $e(g, g)^{xyz}$ on inputs $\mathbb{G}_1$, $\mathbb{G}_2$, e, g, g^x, g^y, g^z, where $(\mathbb{G}_1, \mathbb{G}_2, e)$ is the output of $\mathcal{IG}$ for a sufficiently large security parameter K, g is a random generator of $\mathbb{G}_1$, and x, y, z are random elements of $\mathbb{Z}_q^*$. The BDH assumption is that $Adv_{\mathcal{IG}}(\mathcal{B})$ is negligible for any efficient $\mathcal{B}$.

III. An Efficient Privacy Preserving Keyword Search Scheme

A. Definitions

Suppose a user U is about to store an encrypted email with keywords $W_1, \ldots, W_k$ on a service provider S, where $k \in \mathbb{Z}^+$. Keywords may be words in headline or accepted date, and k is relatively small. U sends the following message to S:

$$MSG_{U2S} = [E_1(U_{pub}, S_{pub}, m), E_2(U_{pub}, W_1), \ldots, E_2(U_{pub}, W_k)]$$

where U_{pub} is U's public key, S_{pub} is S's public key, and m is the email message body. E_1 and E_2 are public key encryption algorithms. U encrypts the email message body using his own public key U_{pub}, the service provider's public key S_{pub}, and E_1. And then U encrypts keywords $W_1, \ldots, W_k$ using his own public key U_{pub} and E_2. Finally, U appends to the encrypted email message body with all the encrypted keywords and sends MSG_{U2S} to S.

Our goal is to enable U to send a trapdoor T_W for a certain keyword W which is encrypted under his private key to S, which will enable S to find out all emails containing the keyword W, but learns nothing else. S then participates in the partial decipherment to calculate an intermediate result of the decipherment using its private key before returning the relevant encrypted emails back. We formally define the efficient privacy preserving keyword search scheme (**EPPKS** for short) below. **EPPKS** supports multiple keywords search on the encrypted data. For the sake of illustration, we only show a single keyword search case in this paper.

*Definition 3.1 (**EPPKS**):* **EPPKS** consists of seven randomized polynomial time algorithms as follows:

1) **Keygen**: takes a sufficiently large security parameter K_1 as an input, and produces a public/private key pair (U_{pub}, U_{priv}) for a user. We write $Keygen\ (K_1) = (U_{pub}, U_{priv})$. Let K_2 be a sufficiently large security parameter, we write $Keygen\ (K_2) = (S_{pub}, S_{priv})$ for a service provider.
2) **EMBEnc**: is a public key encryption algorithm that takes two public keys U_{pub} and S_{pub}, and a message $m \in M$ as inputs, and produces m's cipher-text $C_m \in C_M$. We write $EMBEnc\ (U_{pub}, S_{pub}, m) = C_m$.
3) **KWEnc**: is a public key encryption algorithm that takes a public key U_{pub}, and a keyword $W_i \in W$ ($i \in \mathbb{Z}^+$) as inputs, and produces W_i's cipher-text $C_{W_i} \in C_W$. We write $KWEnc\ (U_{pub}, W_i) = C_{W_i}$.
4) **TCompute**: takes a private key U_{priv} and a keyword W_j ($j \in \mathbb{Z}^+$) as inputs, and produces W_j's trapdoor T_{W_j}. We write $TCompute\ (U_{priv}, W_j) = T_{W_j}$.
5) **Test**: takes a public key U_{pub}, an encrypted keyword C_{W_i}, and a trapdoor T_{W_j} as inputs, and outputs 1 or 0. We write $Test\ (U_{pub}, C_{W_i}, T_{W_j}) = 1$ if $W_i = W_j$, and 0 otherwise.
6) **Decrypt**: takes a private key S_{priv}, a public key U_{pub}, and a cipher-text C_m as inputs, and outputs an intermediate result C_ρ. We write $Decrypt\ (S_{priv}, U_{pub}, C_m) = C_\rho$.
7) **Recovery**: takes a private key U_{priv}, a cipher-text C_m, and an intermediate result C_ρ as inputs, and outputs the plain-text m. We write $Recovery\ (U_{priv}, C_m, C_\rho) = m$.

The user and the service provider run **Keygen** to generate their public/private key pairs respectively. Given U_{pub} and S_{pub}, U runs **EMBEnc** to encrypt an email message body. Given U_{pub}, U runs **KWEnc** to encrypt keywords respectively. When U wants to retrieve emails containing keyword W_j ($j \in \mathbb{Z}^+$), he runs **TCompute** to generate W_j's trapdoor T_{W_j} and sends it to S. S runs **Test** to determine whether a given email contains keyword W_j specified by U. S runs **Decrypt** to calculate an intermediate result C_ρ of the decipherment, and returns back C_ρ along with the encrypted emails. Given a cipher-text and C_ρ, U runs **Recovery** to recover the plain-text.

B. Semantic Security of the *EPPKS* Scheme

In this section, we define security for the **EPPKS** scheme in the sense of semantic security. Semantic security captures our intuition that given a cipher-text the adversary learns nothing about the corresponding plain-text, thus we also say that a semantically secure scheme is IND-CPA secure [8]. According to the definition of **EPPKS**, it consists of two public key encryption algorithms, i.e., **KWEnc** and **EMBEnc**. Therefore, we first define semantic security for **EMBEnc** and **KWEnc**, and then give our definition of a semantically secure **EPPKS** scheme.

*Definition 3.2 (Semantic Security of **KWEnc**):* Given a public key encryption algorithm **KWEnc** which encrypts keywords using U_{pub}, let $\mathcal{A}_1$ be a polynomial time IND-CPA adversary that can adaptively ask for the trapdoor T_{W_i} for any keyword $W_i \in W$ of its choice. $\mathcal{A}_1$ first chooses two keywords W_0 and W_1, which are not to be asked for trapdoors previously, and sends them to **KWEnc**. And then **KWEnc** picks a random element $b_1 \in \{0, 1\}$ and gives $\mathcal{A}_1$ the cipher-text $C_{W_{b_1}} = KWEnc\ (U_{pub}, W_{b_1})$. Finally, $\mathcal{A}_1$ outputs a guess $b_1^{'} \in \{0, 1\}$ for b_1. We define the advantage of $\mathcal{A}_1$ in breaking **KWEnc** as $Adv_{\mathcal{A}_1}(k) = |Pr[b_1 = b_1^{'}] - \frac{1}{2}|$. We say that **KWEnc** is semantically secure if for any polynomial time adversary $\mathcal{A}_1$, the function $Adv_{\mathcal{A}_1}(k)$ is negligible.

*Definition 3.3 (Semantic Security of **EMBEnc**):* Given a public key encryption algorithm **EMBEnc** which encrypts the email message body using U_{pub} and S_{pub}, let $\mathcal{A}_2$ be a polynomial time IND-CPA adversary that can adaptively ask for the cipher-text for any message $m_i \in M$ of its choice. $\mathcal{A}_2$ first chooses two messages m_0 and m_1, which are not to be asked for the cipher-text previously, and sends them to **EMBEnc**. And then **EMBEnc** picks a random $b_2 \in \{0, 1\}$ and gives $\mathcal{A}_2$ the cipher-text $C_{m_{b_2}} = EMBEnc\ (U_{pub}, S_{pub}, m_{b_2})$. Finally, $\mathcal{A}_2$ outputs a guess $b_2^{'} \in \{0, 1\}$ for b_2. We define the advantage of $\mathcal{A}_2$ in breaking **EMBEnc** as $Adv_{\mathcal{A}_2}(k) = |Pr[b_2 = b_2^{'}] - \frac{1}{2}|$. We say that **EMBEnc** is semantically secure if for any polynomial time adversary $\mathcal{A}_2$, the function $Adv_{\mathcal{A}_2}(k)$ is negligible.

*Definition 3.4 (Semantic Security of **EPPKS**):* Given an **EPPKS** scheme consisting of **KWEnc** and **EMBEnc**, it takes a security parameter K as input and runs the key generation algorithm **Keygen** to generate the public/private key pairs (U_{pub}, U_{priv}) and (S_{pub}, S_{priv}). Given an adversary $\mathcal{A}$ consisting of two polynomial time algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$, $\mathcal{A}_1$ initiates attacks on **KWEnc** and $\mathcal{A}_2$ initiates attacks on **EMBEnc**. We say that the **EPPKS** Scheme is semantically secure if for any adversary $\mathcal{A}$, the function $Adv_{\mathcal{A}}(k) = Adv_{\mathcal{A}_1}(k) + Adv_{\mathcal{A}_2}(k)$ is negligible.

IV. Construction

A. Construction Based on Bilinear Maps

Boneh et al [8] use bilinear maps on elliptic curves to build an efficient identity-based encryption (IBE) system. We construct **EPPKS** and prove that it is semantically secure, which closely follows that in Boneh et al [8]. Our construction is based on bilinear maps. The security of this scheme is based on the BDH assumption. Let $\mathcal{IG}$ be some BDH parameter generator. We present our scheme by describing the following seven algorithms.

1) **Keygen**: Given a sufficiently large security parameter $K \in \mathbb{Z}^+$, it runs $\mathcal{IG}$ to generate a prime q, two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order q, and a bilinear map e: $\mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$, where g is a generator of $\mathbb{G}_1$. Then it chooses two hash functions H_1, H_3: $\{0,1\}^* \rightarrow \mathbb{G}_1^*$, a hash function H_2: $\mathbb{G}_2 \rightarrow \{0,1\}^{log^q}$, and a hash function H_4: $\mathbb{G}_2 \rightarrow \{0,1\}^n$ for some n, where H_1, H_2, H_3, and H_4 are random oracles. Finally, it picks two random elements $x, y \in \mathbb{Z}_q^*$ and computes g^x and g^y. The plain-text space includes $M \in \{0,1\}^n$ and $W \in \{0,1\}^*$. The cipher-text space includes $C_M = \mathbb{G}_1^* \times \{0,1\}^n$ and $C_W \in \mathbb{G}_2$. The user's public key is g^x with the corresponding private key x. The service provider's public key is g^y with the corresponding private key y.
2) **EMBEnc**: To encrypt the email message body m under a user's public key g^x and a service provider's public key g^y, it picks a random element $r \in \mathbb{Z}_q^*$ and a random element $\rho \in \{0,1\}^n$, computes $u_1 = g^r$, $u_2 = \rho \oplus H_4(e(g^x, g^y)^r)$, $u_3 = m \oplus H_4(e(H_3(\rho), g^x)^r)$, and sets the cipher-text $C_m = (u_1, u_2, u_3)$.
3) **KWEnc**: To encrypt m's keywords $W_1, \ldots, W_k$ ($k \in \mathbb{Z}^+$) under a user's public key g^x, it computes $H_2(e(g^x, H_1(W_i))^r)$, where $W_i \in \{W_1, \ldots, W_k\}$, sets the cipher-text $C_{W_i} = H_2(e(g^x, H_1(W_i))^r)$, and sends the following message to the service provider:

$$MSG'_{U2S} = [C_m, C_{W_1}, \ldots, C_{W_k}]$$

4) **TCompute**: To retrieve only emails containing keyword W_j ($j \in \mathbb{Z}^+$), it computes the trapdoor $T_{W_j} = H_1(W_j)^x \in \mathbb{G}_1$ under a user's private key x, and sends it to the service provider.
5) **Test**: To determine whether a given email contains keyword W_j, it tests whether $C_{W_i} = H_2(e(u_1, T_{W_j})) = H_2(e(g^r, T_{W_j}))$. If so, $Test(U_{pub}, C_{W_i}, T_{W_j})$ outputs 1. Otherwise, it outputs 0. Note that: If W_i=W_j, then $C_{W_i} = H_2(e(g^x, H_1(W_i))^r) = H_2(g^r, H_1(W_j)^x)) = H_2(e(g^r, T_{W_j})) = H_2(e(u_1, T_{W_j}))$ as required.
6) **Decrypt**: To get an intermediate result of the partial decipherment, it calculates ρ, computes $C_\rho = e(H_3(\rho), u_1) = e(H_3(\rho), g^r)$, and sends the following results to the user:

$$MSG_{S2U} = [C_m, C_{W_1}, \ldots, C_{W_k}, C_\rho]$$

Note that: $\rho = u_2 \oplus H_4(e(g^x, g^y)^r) = u_2 \oplus H_4(e(g^x, g^r)^y)$. Therefore, it could calculate ρ using a service provider's private key y.
7) **Recovery**: Given the cipher-text $C_m = (u_1, u_2, u_3)$ and C_ρ, it computes $m = u_3 \oplus H_4((C_\rho)^x)$ to recover the message m. Note that: $m = u_3 \oplus H_4(e(H_3(\rho), g^x)^r) = u_3 \oplus H_4(e(H_3(\rho), g^r)^x) = u_3 \oplus H_4((C_\rho)^x)$.

For the sake of reducing the computational overhead and increasing the search speed, a service provider could calculate ρ as soon as it receives MSG'_{U2S}, and stores the message as follows:

$$MSG_{Stored@S} = [C_m, C_{W_1}, \ldots, C_{W_k}, \rho]$$

According to the **PEKS** scheme proposed in Boneh et al [7], a user encrypts the email message body using a standard public key system, and a service provider simply returns the relevant emails back after finishing the search. Our **EPPKS** scheme enables a service provider to participate in the partial decipherment to get an intermediate result of the decipherment before returning back the search results, but has no ability to recover the plain-text, which will reduce the computational overhead of the client greatly. Now, we show how to implement these improvements. In the next section, we will give a detailed proof of **EPPKS**.

According to **EPPKS**, a service provider is able to calculate ρ using its private key y. Suppose the service provider knows $H_3(\rho) = g^a \in \mathbb{G}_1$ where a is a random element in $\mathbb{Z}_q^*$, $u_1 = g^r \in \mathbb{G}_1$ where $r \in \mathbb{Z}_q^*$ is a random element chosen by the user, and the user's public key $g^x \in \mathbb{G}_1$ where $x \in \mathbb{Z}_q^*$ is the user's private key, it couldn't calculate $e(g,g)^{arx}$, assuming the BDH problem is hard. In other words, a service provider needs to compute $m = u_3 \oplus H_4(e(H_3(\rho), g^x)^r) = u_3 \oplus H_4(e(g,g)^{arx})$ to recover the plain-text, which corresponds to computing the BDH problem. Therefore, the service provider, seeing only a random value ρ and calculating an intermediate result of the decipherment, has no idea what the plain-text is. Furthermore, an outer attacker couldn't calculate ρ if he doesn't capture any private key. Suppose an outer attacker knows $u_1 = g^r$ by intercepting MSG'_{U2S}, the user's public key $g^x \in \mathbb{G}_1$, and the service provider's public key $g^y \in \mathbb{G}_1$, it couldn't calculate $e(g,g)^{xyr}$, assuming the BDH problem is hard. In other words, an outer attacker needs to compute $\rho = u_2 \oplus H_4(e(g^x, g^y)^r) = u_2 \oplus H_4(e(g,g)^{xyr})$ to recover ρ, which corresponds to computing the BDH problem.

B. Security

In this section, we study the security of the proposed **EPPKS** scheme. The following theorem shows that **EPPKS** is semantically secure if the BDH problem is assumed to be hard.

Theorem 4.1: Suppose the hash functions H_1, H_2, H_3, and H_4 are random oracles. Then **EPPKS** is semantically secure assuming the BDH problem is hard. Let $\mathcal{A}$ be an IND-CPA adversary consisting of two polynomial time algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$. Let $\mathcal{A}_1$ be an IND-CPA adversary that has the advantage ϵ_1 in breaking **KWEnc**. Suppose $\mathcal{A}_1$ makes $q_T > 0$ trapdoor queries and $q_{H_2} > 0$ hash queries to H_2. Let $\mathcal{A}_2$ be an IND-CPA adversary that has the advantage ϵ_2 against **EMBEnc**. Suppose $\mathcal{A}_2$ makes $q_{H_4} > 0$ hash function queries to H_4. Let $\mathcal{A}$ be an IND-CPA adversary that has the advantage $\epsilon = \epsilon_1 + \epsilon_2$ against the **EPPKS** scheme. Then there is an algorithm $\mathcal{B}$ that solves the BDH problem with the advantage at least:

$$Adv_{\mathcal{B}}(K) \geq 2\epsilon_1 / \{e \cdot q_{H_2} \cdot (1 + q_T)\} + 2\epsilon_2 / q_{H_4} \qquad (1)$$

Here $e \approx 2.71$ is the base of the natural logarithm. The running time of $\mathcal{B}$ is $O(time(\mathcal{A}))$.

EPPKS includes two public key encryption algorithms, i.e., **EMBEnc** and **KWEnc**. Therefore, we prove Theorem

4.1 in two steps. We first show that **KWEnc** is semantically secure if the BDH assumption holds.

Lemma 4.2: Let H_1 be a random oracle from $\{0,1\}^*$ to $\mathbb{G}_1^*$ and H_2 be a random oracle from $\mathbb{G}_2$ to $\{0,1\}^{log^q}$. Suppose $\mathcal{A}_1$ be an IND-CPA adversary that has the advantage ϵ_1 in breaking **KWEnc**. Suppose $\mathcal{A}_1$ makes at most $q_{H_2} > 0$ hash queries to H_2 and at most $q_T > 0$ trapdoor queries. Then there is an algorithm $\mathcal{B}_1$ that solves the BDH problem with the advantage at least $\epsilon_1^{'} = 2\epsilon_1/\{e \cdot q_{H_2} \cdot (1+q_T)\}$, and a running time $O(time(\mathcal{A}_1))$.

Proof. Let $< q, \mathbb{G}_1, \mathbb{G}_2, e >$ be the BDH parameters, where q is the prime order of $\mathbb{G}_1$ and $\mathbb{G}_2$. Choose a random generator $g \in \mathbb{G}_1$. $\mathcal{B}_1$ is given $\upsilon_0 = g$, $\upsilon_1 = g^{\alpha_1}$, $\upsilon_2 = g^{\beta_1}$, $\upsilon_3 = g^{\gamma_1} \in \mathbb{G}_1$ where α_1, β_1, γ_1 are random elements in $\mathbb{Z}_q^*$. Its goal is to output $D_1 = e(g,g)^{\alpha_1\beta_1\gamma_1} \in \mathbb{G}_2$. Let D_1 be the solution to the BDH problem. $\mathcal{B}_1$ finds D_1 by interacting with $\mathcal{A}_1$ as follows:

Keygen: $\mathcal{B}_1$ sends (υ_0, υ_1) as the public key to $\mathcal{A}_1$.

H_1-Queries: $\mathcal{B}_1$ maintains a list of tuples called H_1-List, in which each entry is a tuple of the form $< W_j, h_j, a_j, c_j >$. The list is initially empty. When $\mathcal{A}_1$ queries the random oracle H_1 at a point $W_i \in \{0,1\}^*$, $\mathcal{B}_1$ responds as follows:

1) If W_i already appears on H_1-List in a tuple $< W_i, h_i, a_i, c_i >$, then $\mathcal{B}_1$ responds with $H_1(W_i) = h_i \in \mathbb{G}_1^*$.
2) Otherwise, $\mathcal{B}_1$ generates a random $coin \in \{0,1\}$, so that $Pr[coin = 0] = \delta$ for some δ that will be determined later.
3) $\mathcal{B}_1$ picks a random $a \in \mathbb{Z}_q^*$. If $coin = 0$, $\mathcal{B}_1$ computes $h_i = \upsilon_2 \cdot g^a = g^{\beta_1} \cdot g^a \in \mathbb{G}_1^*$. If $coin = 1$, $\mathcal{B}_1$ computes $h_i = g^a \in \mathbb{G}_1^*$.
4) $\mathcal{B}_1$ adds the tuple $< W_i, h_i, a, coin >$ to H_1-List and responds to $\mathcal{A}_1$ with $H_1(W_i) = h_i$. Note that either way h_i is uniform in $\mathbb{G}_1^*$ and is independent of $\mathcal{A}_1$'s current view as required.

H_2-Queries: $\mathcal{B}_1$ maintains a list of tuples called H_2-List, in which each entry is a tuple of the form $< t_j, v_j >$. The list is initially empty. When $\mathcal{A}_1$ issues a query to H_2, $\mathcal{B}_1$ checks if t_i is already on H_2-List in the form of $< t_i, v_i >$. If so, $\mathcal{B}_1$ responds to $\mathcal{A}_1$ with $H_2(t_i) = v_i$. Otherwise, $\mathcal{B}_1$ picks a random string $v_i \in \{0,1\}^{log^q}$, adds the tuple $< t_i, v_i >$ to H_2-List, and responds to $\mathcal{A}_1$ with $H_2(t_i) = v_i$.

Phase 1: When $\mathcal{A}_1$ issues a query for the trapdoor of keyword W_i, $\mathcal{B}_1$ responds as follows:

1) $\mathcal{B}_1$ initiates H_1-Queries to obtain $h_i \in \mathbb{G}_1^*$, where $H_1(W_i) = h_i$. Let $< W_i, h_i, a_i, c_i >$ be the corresponding tuple on H_1-List. If $c_i = 0$, then $\mathcal{B}_1$ reports a failure and terminates.
2) If $c_i = 1$, then $H_1(W_i) = h_i = g^{a_i} \in \mathbb{G}_1^*$. We define $T_{W_i} = (\upsilon_1)^{a_i} = (g^{\alpha_1})^{a_i}$. Note that $T_{W_i} = (g^{\alpha_1})^{a_i} = (g^{a_i})^{\alpha_1} = H_1(W_i)^{\alpha_1}$. $\mathcal{B}_1$ gives T_{W_i} to $\mathcal{A}_1$.

Challenge: Once $\mathcal{A}_1$ decides that **Phase 1** is over, it outputs a pair of keywords W_0 and W_1 on which it wishes to be challenged. $\mathcal{B}_1$ responds as follows:

1) $\mathcal{B}_1$ initiates H_1-Queries twice to obtain h_0 and $h_1 \in \mathbb{G}_1^*$, where $H_1(W_0) = h_0$ and $H_1(W_1) = h_1$. If $c_0 = 1$ or $c_1 = 1$, then $\mathcal{B}_1$ reports a failure and terminates.
2) If both $c_0 = 0$ and $c_1 = 0$, $\mathcal{B}_1$ randomly picks a $b_1 \in \{0,1\}$.
3) $\mathcal{B}_1$ picks a random string $S_1 \in \{0,1\}^{log^q}$, and gives the cipher-text $C_1 = (\upsilon_3, S_1)$ to $\mathcal{A}_1$. Note that:
$S_1 = H_2(e(\upsilon_1, H_1(W_{b_1}))^{\gamma_1}) = H_2(g^{\alpha_1}, H_1(W_{b_1}))^{\gamma_1})$
$= H_2(e(g^{\alpha_1}, g^{\beta_1} \cdot g^{a_b})^{\gamma_1}) = H_2(e(g,g)^{\alpha_1\gamma_1(\beta_1+a_b)})$
Hence, C_1 is a valid cipher-text for W_{b_1} as required.

Phase 2. $\mathcal{A}_1$ can continue issuing more trapdoor queries for keyword W_i, where the only restriction is that $W_i \neq W_0$ and $W_i \neq W_1$. $\mathcal{B}_1$ responds as in **Phase 1.**

Guess: $\mathcal{A}_1$ outputs its guess $b_1^{'} \in \{0,1\}$ for b_1. $\mathcal{B}_1$ picks a random pair $< t_i, v_i >$ from H_2-List and outputs t_i as the solution to D_1.

To complete the proof of Lemma 4.2, we now show that $\mathcal{B}_1$ correctly outputs D_1 with the probability at least $\epsilon_1^{'} = 2\epsilon_1/\{e \cdot q_{H_2} \cdot (1+q_T)\}$. In the first place, we calculate the probability that $\mathcal{B}_1$ does not abort during the above process. Suppose $\mathcal{A}_1$ makes a total of q_T trapdoor queries. Then the probability $\mathcal{B}_1$ does not abort in **Phase 1** or **2** is δ^{q_T}. And the probability that it does not abort during the challenge step is $1-\delta$. Therefore, the probability that $\mathcal{B}_1$ does not abort during the whole process is $\delta^{q_T} \cdot (1-\delta)$. This value is maximized at $\delta_{opt} = 1 - 1/(q_T+1)$. Using δ_{opt}, the probability that $\mathcal{B}_1$ does not abort is at least $1/e(1+q_T)$. In the second place, we calculate the probability that $\mathcal{B}_1$ outputs the correct result in case that $\mathcal{B}_1$ does not abort. Let Q_1 be the event that $\mathcal{A}_1$ issues a query for v. If $\neg Q_1$, we know that the decryption of the cipher-text is independent of $\mathcal{A}_1$'s view. Let $Pr[b_1 = b_1^{'}]$ be the probability that $\mathcal{A}_1$ outputs the correct result, therefore in the real attack $Pr[b_1 = b_1^{'} | \neg Q_1] = \frac{1}{2}$. Since $\mathcal{A}_1$ has the advantage ϵ_1, $|Pr[b_1 = b_1^{'} | \neg Q_1] - \frac{1}{2}| \geq \epsilon_1$. According to the following formulae, we know $Pr[Q_1] \geq 2\epsilon_1$.

$$\begin{aligned} Pr[b_1 = b_1^{'}] &= Pr[b_1 = b_1^{'} | \neg Q_1] Pr[\neg Q_1] \\ &+ Pr[b_1 = b_1^{'} | Q_1] Pr[Q_1] \\ &\leq \tfrac{1}{2} Pr[\neg Q_1] + Pr[Q_1] \\ &= \tfrac{1}{2} + \tfrac{1}{2} Pr[Q_1] \\ Pr[b_1 = b_1^{'}] &\geq Pr[b_1 = b_1^{'} | \neg Q_1] Pr[\neg Q_1] \\ &= \tfrac{1}{2} Pr[\neg Q_1] \\ &= \tfrac{1}{2} - \tfrac{1}{2} Pr[Q_1] \end{aligned}$$

Therefore, we have that $Pr[Q_1] \geq 2\epsilon_1$ in the real attack. Now we know that $\mathcal{A}_1$ will issue a query for v with the probability at least $2\epsilon_1$. That is to say, the probability that v appears in some pair on H_2-List is at least $2\epsilon_1$. $\mathcal{B}_1$ will choose the correct pair with the probability at least $1/q_{H_2}$ and thus $\mathcal{B}_1$ produces the correct answer with the probability at least $2\epsilon_1/q_{H_2}$. Since $\mathcal{B}_1$ does not abort with the probability at least $1/e(1+q_T)$, we see that $\mathcal{B}_1$'s success probability is at least $\epsilon_1^{'} = 2\epsilon_1/\{e \cdot q_{H_2} \cdot (1+q_T)\}$ as required.

Next, we show that **EMBEnc** is a semantically secure public key encryption if the BDH assumption holds. It is worth noticing that the outer attackers couldn't calculate ρ if the BDH assumption holds. Without loss of generality, we suppose that an IND-CPA adversary $\mathcal{A}_2$ has already known ρ and could issue H_3 queries at any time.

Lemma 4.3: Let H_3 be a random oracle from $\{0,1\}^*$ to

$\mathbb{G}_1^*$ and H_4 be a random oracle from $\mathbb{G}_2$ to $\{0,1\}^n$. Let $\mathcal{A}_2$ be an IND-CPA adversary that has the advantage ϵ_2 against **EMBEnc**. Suppose $\mathcal{A}_2$ makes $q_{H_4} > 0$ hash function queries to H_4. Then there is an algorithm $\mathcal{B}_2$ that solves the BDH problem with the advantage at least $\epsilon_2^{'} = 2\epsilon_2/q_{H_4}$ and a running time $O(time(\mathcal{A}_2))$.

Proof. $\mathcal{B}_2$ is given $\rho \in \{0,1\}^n$, $\mu_0 = g$, $\mu_1 = g^{\alpha_2}$, $\mu_2 = g^{\beta_2}$, $\mu_3 = g^{\gamma_2} \in \mathbb{G}_1$, where α_2, β_2, γ_2 are random elements in $\mathbb{Z}_q^*$. Its goal is to output $D_2 = e(g,g)^{\alpha_2\beta_2\gamma_2} \in \mathbb{G}_2$. Let D_2 be the solution to the BDH problem. $\mathcal{B}_2$ finds D_2 by interacting with $\mathcal{A}_2$ as follows:

Keygen: $\mathcal{B}_2$ sends (μ_0, μ_1) as the public key to $\mathcal{A}_2$.

H_3-Queries: $\mathcal{B}_2$ maintains a list of tuples called H_3-List, in which each entry is a tuple of the form $< \rho_j, f_j >$. The list is initially empty. When $\mathcal{A}_2$ issues a query to H_3, $\mathcal{B}_2$ checks if ρ_i is already on H_3-List in the form of $< \rho_i, f_i >$. If so, $\mathcal{B}_2$ responds to $\mathcal{A}_2$ with $H_3(\rho_i) = f_i$. Otherwise, $\mathcal{B}_2$ picks a random $d \in \mathbb{Z}_q^*$, computes $f_i = \mu_2 \cdot g^d = g^{\beta_2} \cdot g^d \in \mathbb{G}_1^*$, adds the tuple $< \rho_i, f_i >$ to H_3-List, and responds to $\mathcal{A}_2$ with $H_3(\rho_i) = f_i$.

H_4-Queries: $\mathcal{B}_2$ maintains a list of tuples called H_4-List, in which each entry is a tuple of the form $< r_j, l_j >$. The list is initially empty. When $\mathcal{A}_2$ issues a query to H_4, $\mathcal{B}_2$ checks if r_i is already on H_4-List in the form of $< r_i, l_i >$. If so, $\mathcal{B}_2$ responds to $\mathcal{A}_2$ with $H_4(r_i) = l_i$. Otherwise, $\mathcal{B}_2$ picks a random string $l_i \in \{0,1\}^n$, adds the tuple $< r_i, l_i >$ to H_4-List, and responds to $\mathcal{A}_2$ with $H_4(r_i) = l_i$.

Challenge. $\mathcal{A}_2$ outputs two messages m_0 and m_1 on which it wishes to be challenged. $\mathcal{B}_2$ randomly picks $b_2 \in \{0,1\}$ and a random string $S_2 \in \{0,1\}^n$, and gives the cipher-text $C_2 = (\mu_3, S_2)$ to $\mathcal{A}_2$. Note that the decryption of the cipher-text is:

$m_{b_2} = S_2 \oplus H_4(e(H_3(\rho), \mu_1)^{\gamma_2}) = S_2 \oplus H_4(e(H_3(\rho), g^{\alpha_2})^{\gamma_2})$
$= S_2 \oplus H_4(e(g^{\beta_2} \cdot g^d, g^{\alpha_2})^{\gamma_2}) = S_2 \oplus H_4(e(g,g)^{\alpha_2\gamma_2(\beta_2+d)})$

Hence, C_2 is a valid cipher-text for m_{b_2} as required.

Guess: $\mathcal{A}_2$ outputs its guess $b_2^{'} \in \{0,1\}$ for b_2. $\mathcal{B}_2$ picks a random pair $< r_i, l_i >$ from H_4-List and outputs r_i as the solution to the given instance of BDH.

Let Q_2 be the event that $\mathcal{A}_2$ issues a query for l. From proof of Lemma 4.2, we know that $Pr[Q_2] \geq 2\epsilon_2$. That is to say, $\mathcal{A}_2$ will issue a query for l with the probability at least $2\epsilon_2$. $\mathcal{B}_2$ will choose the correct pair with the probability at least $1/q_{H_4}$ and thus $\mathcal{B}_2$ produces the correct answer with the probability at least $\epsilon_2^{'} = 2\epsilon_2/q_{H_4}$ as required.

Proof of Theorem 4.1. The theorem follows directly from Lemma 4.2 and Lemma 4.3. It shows that an IND-CPA adversary $\mathcal{A}$ on **EPPKS** with the advantage ϵ gives a BDH algorithm with the advantage at least $Adv_{\mathcal{B}} \geq 2\epsilon_1/e \cdot q_{H_2} \cdot (1 + q_T) + 2\epsilon_2/q_{H_4}$ as required.

V. Conclusion

Cloud computing is one of the current most important and promising technologies. A user could store his personal files in a cloud and retrieves them wherever and whenever he wants. For the sake of protecting the user data privacy and the user queries privacy, we propose an efficient privacy preserving keyword search scheme in cloud computing. It allows a service provider to participate in partial decipherment, thus the user could pay less computational overhead for decryption. Furthermore, it is a searchable encryption scheme, thus the service provider could search the encrypted files efficiently without leaking any information. By proof, it is semantically secure.

In many cases, the user might want the service provider not only to provide the storage service, but also to provide the computational service. Ideally, the service provider has the ability to output the right answers but knows nothing about the data of the user. In other words, the service provider needs to compute on the encrypted data, which is a big challenge for us. In our future work, we will dedicate to this research direction.

Acknowledgments

This work is supported by the National Natural Science Foundation of China under Grant Nos. 90718034 and 60773013, the Program for New Century Excellent Talents in University (NCET-06-0686), and the Program for Changjiang Scholars and Innovative Research Team in University under Grant No. IRT0661.

References

[1] A. Weiss. Computing in the Clouds. netWorker, 11(4):16-25, Dec. 2007.
[2] Gartner Identifies the Top 10 Strategic Technologies for 2009, http://www.gartner.com/it/page.jsp?id=777212/ [14 Oct 2008].
[3] Amazon Elastic Compute Cloud (EC2), http://www.amazon.com/ec2/ [18 Jul 2008].
[4] Google App Engine, http://appengine.google.com [18 Jul 2008].
[5] Microsoft Live Mesh, http://www.mesh.com [18 Jul 2008].
[6] H. Haclgiimfi, B. Iyer, C. Li, and S. Mehrotra. Executing SQL over Encrypted Data in Database-Service-Provider Model. Technical Report TR-DB-02-02, Database Research Group at University of California, Irvine, 2002.
[7] D. Boneh, G. Crescenzo, R. Ostrovsky, and G. Persiano. Public Key Encryption with Keyword Search. Proceedings of Eurocrypt 2004, Lecture Notes in Computer Science 3027, pp. 506-522.
[8] D. Boneh and M. Franklin. Identity Based Encryption from the Weil Pairing. SIAM J. of Computing, 32 (3): 586-615, 2003, also as an extended abstract in Crypto 2001.
[9] D. X. Song, D. Wagner and A. Perrig. Practical Techniques for Searches on Encrypted Data. Proceedings of the 2000 IEEE Symposium on Security and Privacy, pp. 44-55.
[10] A. K. Bennett, C. Grothoff, T. Horozov, and I. Patrascu. Efficient Sharing of Encrypted Data. Proceedings of ACISP 2002, Lecture Notes in Computer Science 2384, pp. 107-120.
[11] T. Diament, H. K. Lee, A. D. Keromytis and M. Yung. The Dual Receiver Cryptosystem and its Applications. Proceedings of the ACM CCS 2004, pp. 330-343.
[12] Y.-C. Chang and M. Mitzenmacher. Privacy Preserving Keyword Searches on Remote Encrypted Data. Proceedings of ACSN 2005, Lecture Notes in Computer Science 3531, pp. 442-455.
[13] M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval. Key-Privacy in Public-Key Encryption. Proceedings of Advances in Cryptology - Asiacrypt 2001, Lecture Notes in Computer Science 2248, pp. 566-582.
[14] K. Ren, W. Lou, K. Kim, and R. Deng. A Novel Privacy Preserving Authentication and Access Control Scheme for Pervasive Computing Environment. IEEE Transactions on Vehicular Technology, 55(4):1373-1384, Jul. 2006.
[15] D. Boneh and B. Waters. Conjunctive, Subset, and Range Queries on Encrypted Data. Proceedings of TCC 2007, Lecture Notes in Computer Science 4392, pp. 535-554.
[16] E. Shi, J. Bethencourt, T-H. H. Chan, D. Song, and A. Perrig. Multi-Dimensional Range Query over Encrypted Data. Proceedings of IEEE Symposium on Security and Privacy 2007, pp. 350-364.