Journal of Network and Computer Applications ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

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

Secure and privacy preserving keyword searching for cloud storage services

Qin Liu [a,b], Guojun Wang [a,*], Jie Wu [b]

[a] School of Information Science and Engineering, Central South University, Changsha, Hunan Province 410083, PR China
[b] Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122, USA

ARTICLE INFO

Article history:
Received 29 July 2010
Received in revised form
11 January 2011
Accepted 9 March 2011

Keywords:
Cloud storage
Security
Privacy preserving
Partial decipherment
Searchable encryption

ABSTRACT

Cloud storage services enable users to remotely access data in a cloud anytime and anywhere, using any device, in a pay-as-you-go manner. Moving data into a cloud offers great convenience to users since they do not have to care about the large capital investment in both the deployment and management of the hardware infrastructures. However, allowing a cloud service provider (CSP), whose purpose is mainly for making a profit, to take the custody of sensitive data, raises underlying security and privacy issues. To keep user data confidential against an untrusted CSP, a natural way is to apply cryptographic approaches, by disclosing the data decryption key only to authorized users. However, when a user wants to retrieve files containing certain keywords using a thin client, the adopted encryption system should not only support keyword searching over encrypted data, but also provide high performance. In this paper, we investigate the characteristics of cloud storage services and propose a secure and privacy preserving keyword searching (SPKS) scheme, which allows the CSP to participate in the decipherment, and to return only files containing certain keywords specified by the users, so as to reduce both the computational and communication overhead in decryption for users, on the condition of preserving user data privacy and user querying privacy. Performance analysis shows that the SPKS scheme is applicable to a cloud environment.

1. Introduction

Cloud computing (Weiss, 2007), as one of the 2010 top 10 strategic technologies, dynamically provides high-quality cloud-based services and applications over the Internet. Cloud storage services, which can be regarded as a kind of typical service in cloud computing, involves the delivery of data storage as a service. Cloud storage services, including database-like services and network attached storage, are often billed on a utility computing basis, e.g., per gigabyte per month. For example, the Amazon simple storage service (Amazon S3) charges only from $0.12 to $0.15 per gigabyte per month.

One of the biggest merits of cloud storage is that users can access data in a cloud anytime and anywhere, using any device. We consider the following application scenario: A user *U* pays a cloud service provider (CSP) for a cloud 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 or a mobile phone.

It is trivial to do so when the email messages are stored in the form of plaintexts. But, this will result in undesirable security and privacy risks. For example, *U* is a technician in Company A who is in charge of after-sale services. He stores all the emails sent from customers in a cloud when he is in the office with a desktop, and retrieves them to tackle service requests from customers when he is out in the field with his PDA. In such an environment, an attacker who intercepts and captures the communications is able to know a customer's privacy information as well as some important business secrets. Even worse, an untrustworthy CSP is able to easily obtain all the information and sell it to the biggest rival of Company A.

As described in Haclgiimfi et al. (2002), there are two main attacks under such a circumstance, i.e., external attacks initiated by unauthorized outsiders and internal attacks initiated by untrustworthy CSPs. In some cases, we cannot fully trust a CSP, but still need its services. Therefore, some mechanisms are needed to protect the user data privacy and the user querying privacy in a cloud environment. To protect user querying privacy is equally important in user data privacy. Without proper protection for the user querying privacy, an attacker may know the user's private interests and querying patterns. For the above example, Company B is able to know who sends emails more often than other customers by analyzing the frequency of occurrences of the keywords, which are the customers' email addresses, and classifies them as its potential customers.

The natural approach is to encrypt the emails before storing them in the cloud and send queries in the form of encrypted

* Corresponding author.
E-mail addresses: csgjwang@gmail.com, csgjwang@mail.csu.edu.cn (G. Wang).
URL: http://trust.csu.edu.cn/faculty/~csgjwang (G. Wang).

1084-8045/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jnca.2011.03.010

keywords to retrieve them. For example, a user may use his public key to encrypt an email and its keywords before sending it to the CSP, 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 is not aware of the encrypted files, the encrypted keywords, and the user querying 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 CSP cannot determine which emails contain keywords specified by a user if the encryption is not searchable, and can only return all the encrypted emails. Generally speaking, a thin client has only limited bandwidth, CPU, and memory, therefore, a simple encryption scheme cannot work well under these circumstances.

We propose the SPKS scheme for cloud storage services to solve the above problem. Our contributions are threefold:

1. It is efficient and practical. The SPKS scheme enables CSPs to participate in the partial decipherment so as to reduce computational overhead on users, without leaking any information about the plaintext. We analyze the performance of our scheme, and show that it outperforms the scheme proposed by Boneh et al. (2004) when applied to a cloud environment.
2. It supports keyword searching on encrypted data. The SPKS scheme enables the CSP to determine whether a given email contains certain keywords specified by a user, but is not aware of any information about both the keywords and the email.
3. It is a provably secure scheme. The SPKS scheme can be proved to be semantically secure under the Bilinear Diffie–Hellman (BDH) assumption and the random oracle model (Boneh and Franklin, 2003).

This paper is structured as follows: First, we review some related work in Section 2, and introduce some preliminaries in Section 3. Then, we outline the SPKS scheme and give security definition in Section 4. Next, we construct the proposed scheme in Section 5, and analyze its performance in Section 6. Finally, we conclude this paper in Section 7.

2. Related work

The question on how to achieve keyword searching on encrypted data efficiently was first raised in Song et al. (2000). Since then, there has been much work conducted in this field, such as Haclgiimfi et al. (2002), Boneh et al. (2004), Chang and Mitzenmacher (2005), Boneh and Waters (2007), Shi et al. (2007), and Liu et al. (2009). Boneh et al. (2004) proposed a public key encryption with keyword searching (PEKS) scheme, which enables a gateway to test whether certain keywords are contained in an email without learning any information about the email and keywords. The key technique to make the PEKS scheme work is that an email and corresponding keywords are encrypted under the standard public key encryption algorithm and the PEKS algorithm, respectively.

In our previous work (Liu et al., 2009), to allow users to efficiently access files containing certain keywords in a cloud anytime and anywhere using any device, we proposed an efficient privacy preserving keyword searching scheme (EPPKS) in cloud computing by making performance improvements to the PEKS scheme. Inspired by the work of Diament et al. (2004), our previous work introduced the notion of "partial decipherment" into the process of searching encrypted data so as to reduce the computational overhead in decryption for users. The key technique to make the EPPKS scheme work is that a file is encrypted under the public keys of both the CSP and the user, so that the CSP with the ability to calculate an intermediate result of the decipherment using its private key, is able to participate in the partial decipherment before returning files containing certain keywords specified by the user. Based on the BDH assumption, the CSP cannot know file contents and keywords.

However, there are still some unsolved issues in our previous work. First, it lacks performance analysis and comparisons with the existing schemes. Second, a user needs to send the additional n-bits length ciphertext to the CSP for the sake of reducing the computational overhead in decryption, which will increase the communication cost to a certain degree. Third, it assumes that the length of all emails is either shorter than or the same as n, which is not practical in the real environment. In this paper, we propose the SPKS scheme for cloud storage services to address these issues.

3. Preliminaries

In this section, we first introduce some related definitions and complexity assumptions, which closely follows (Boneh and Franklin, 2003), and then outline the PEKS scheme proposed in Boneh et al. (2004).

3.1. Related definitions

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 3.1 (*Bilinear map*). We call $\hat{e}$ a bilinear map if $\hat{e}: \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ is a map with the following properties:

1. Computable: There is a polynomial time algorithm to compute $\hat{e}(g,h) \in \mathbb{G}_2$, for any $g,h \in \mathbb{G}_1$.
2. Bilinear: $\hat{e}(g^x,h^y) = \hat{e}(g,h)^{xy}$ 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 $\hat{e}(g,g)$ is a generator of $\mathbb{G}_2$.

Note that the Weil and Tate pairings associated with supersingular elliptic curves or abelian varieties can be modified to create a bilinear map. In Section 6, we use the Weil pairing to construct a bilinear map to analyze the performance of our scheme.

Definition 3.2 (*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 $\hat{e}: \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$.

Definition 3.3 (*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 $\hat{e}(g,g)^{xyz} \in \mathbb{G}_2$.

Definition 3.4 (*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 as the probability that $\mathcal{B}$ outputs $\hat{e}(g,g)^{xyz}$ on inputs $\mathbb{G}_1$, $\mathbb{G}_2$, $\hat{e}$, g, g^x, g^y, g^z, where $(\mathbb{G}_1,\mathbb{G}_2,\hat{e})$ is the output of $\mathcal{IG}$ for 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 $\mathcal{B}$.

3.2. Outline of the PEKS scheme

The application context of the PEKS scheme 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

Please cite this article as: Liu Q, et al. Secure and privacy preserving keyword searching for cloud storage services. J Network Comput Appl (2011), doi:10.1016/j.jnca.2011.03.010

contains the keyword *urgent* so that it could route the email to her PDA immediately; (3) But, Alice does not want the email gateway to be able to decrypt her messages. The PEKS scheme, which works as in Fig. 1, consists of the following four algorithms:

1. *KeyGen*(K): Takes a security parameter K as input, and generates a public/private key pair (A_{pub},A_{priv}) for Alice.
2. *PEKS*(A_{pub},W') : Given Alice's public key A_{pub} and a word W', produces a searchable encryption S for W'.
3. *Trapdoor*(A_{priv},W): Given Alice's private key A_{priv} and a keyword W, produces a trapdoor TW for W.
4. *Test*(A_{pub},S,TW): Given Alice's public key A_{pub}, a searchable encryption $S = PEKS(A_{pub},W')$, and a trapdoor $TW = Trapdoor(A_{priv},W)$, outputs "yes" if $W = W'$ and "no" otherwise.

The PEKS scheme can be easily applied to the application scenario in Section 1 as follows: (1) Alice and Bob might be the same person, i.e., the user U, who runs the *KeyGen* algorithm to generate his public/private key pair; (2) U first uses the standard public key encryption algorithm and the *PEKS* algorithm to encrypt an email and corresponding keywords, respectively. (3) When U wants to retrieve emails containing a keyword W, he runs the *Trapdoor* algorithm to compute a short trapdoor for W using his private key, and then sends it to the CSP. (4) On receiving the trapdoor, the CSP uses the *Test* algorithm to find all emails containing keyword W while not being aware of emails and W.

However, such a simple transformation may not work well in cloud computing. In the PEKS scheme, a user encrypts the email using a standard public key system, and the CSP simply returns the relevant emails after finishing the search, which enables the user to decrypt the ciphertext by himself. Frequent decryption will deplete too much CPU and memory capabilities of the client and lose the critical virtue of cloud computing, i.e., enabling users to access data in a cloud anytime and anywhere, using any device. Therefore, we propose the SPKS scheme, which enables the CSP not only to determine which files contain certain keywords specified by the user, but also to participate in the partial decipherment to get an intermediate result of the decipherment before returning the search results, so as to reduce both the communication and computational overhead in decryption for the user greatly, on the condition of preserving user data privacy and user querying privacy.

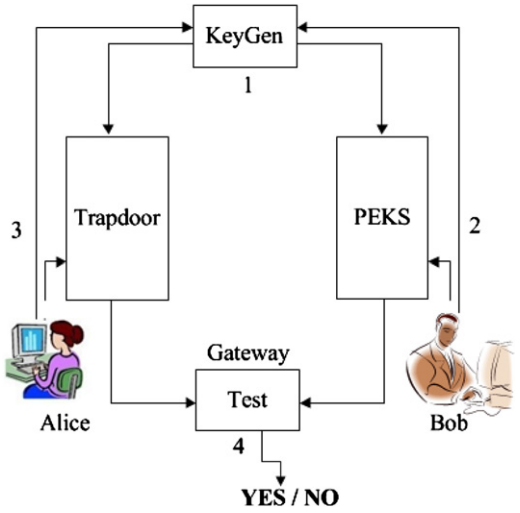

Fig. 1. The working process of the PEKS scheme.

4. Outline of the proposed scheme

4.1. Definitions

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

$$MSG_{U2CSP} = [EMBEnc(U_{pub},S_{pub},m),KWEnc(U_{pub},W_1),\ldots,KWEnc(U_{pub},W_k)],$$

where U_{pub} is U's public key, S_{pub} is the CSP's public key, and m is the email. *EMBEnc* and *KWEnc* are public key encryption algorithms that will be described below.

Definition 4.1 (*SPKS*). The SPKS scheme, which works as in Fig. 2, consists of seven randomized polynomial time algorithms as follows:

1. *KeyGen*: It 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 the CSP.
2. *EMBEnc*: It 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 ciphertext $C_m \in C_M$. We write $EMBEnc(U_{pub},S_{pub},m) = C_m$.
3. *KWEnc*: It 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 ciphertext $C_{W_i} \in C_W$. We write $KWEnc(U_{pub},W_i) = C_{W_i}$.
4. *TCompute*: It 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. *KWTest*: It 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. $KWTest(U_{pub},C_{W_i},T_{W_j}) = 1$ if $W_i = W_j$, and 0 otherwise.
6. *PDecrypt*: It takes a private key S_{priv}, a public key U_{pub}, and a ciphertext C_m as inputs, and outputs an intermediate result C_ρ of the decipherment. We write $PDecrypt(S_{priv},U_{pub},C_m) = C_\rho$.
7. *Recovery*: It takes a private key U_{priv}, a ciphertext C_m, and an intermediate result C_ρ as inputs, and outputs the plaintext m. We write $Recovery(U_{priv},C_m,C_\rho) = m$.

According to Fig. 2, the SPKS scheme works as follows: (1) U and the CSP run the *KeyGen* algorithm to generate their public/private key pairs, respectively. (2) When U wants to store an email m containing keywords $W_1,\ldots,W_k$ on cloud servers, U first

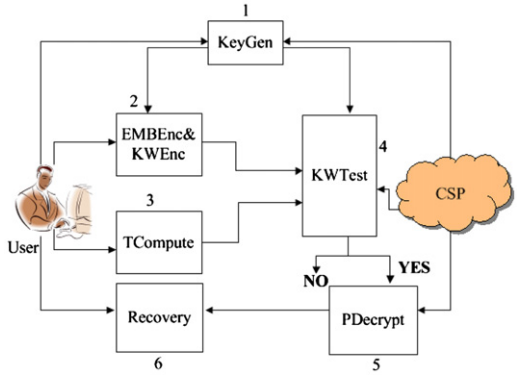

Fig. 2. The working process of the SPKS scheme.

Please cite this article as: Liu Q, et al. Secure and privacy preserving keyword searching for cloud storage services. J Network Comput Appl (2011), doi:10.1016/j.jnca.2011.03.010

runs the *EMBEnc* algorithm to encrypt the email, and then runs *KWEnc* to encrypt all the keywords, respectively, and finally sends both the ciphertext of the email and keywords to the CSP. (3) When U wants to retrieve emails containing keyword $W_j (j \in \mathbb{Z}^+)$, he runs the *TCompute* algorithm to generate W_j's trapdoor T_{W_j} and sends it to the CSP. (4) On receiving the trapdoor, the CSP runs the *KWTest* algorithm to determine whether a given email contains keyword W_j specified by U. (5) Before returning the results to U, the CSP runs *PDecrypt* to calculate an intermediate result C_ρ for the decipherment. After that, it returns C_ρ along with the encrypted emails. (6) Given a ciphertext and C_ρ, U runs the *Recovery* algorithm to recover the plaintext. The SPKS scheme can support multiple keyword searching on the encrypted data while preserving the user privacy. For the sake of illustration, we only show a single keyword searching case in this paper.

4.2. Semantic security of the SPKS scheme

We define security for the SPKS scheme in the sense of semantic security (Boneh and Franklin, 2003). Semantic security captures our intuition that, given a ciphertext, the adversary learns nothing about the corresponding plaintext, thus, we also say that a semantically secure scheme is IND-CPA secure. Recall that the SPKS scheme consists of two public key encryption algorithms, i.e., algorithms *EMBEnc* and *KWEnc*, where *KWEnc* closely follows the PEKS algorithm, whose security had been proved in Boneh et al. (2004). Therefore, we define the semantic security for the SPKS scheme as follows:

Definition 4.2 (*Semantic Security of the SPKS scheme*). Given the SPKS scheme, and two sufficiently large security parameters K_1 and K_2, generate the public/private key pairs $KeyGen(K_1)=(U_{pub},U_{priv})$ and $KeyGen(K_2)=(S_{pub},S_{priv})$. Let $\mathcal{A}$ be a polynomial time IND-CPA adversary that can adaptively ask for the ciphertext for any message $m_i \in M$ of its choice. $\mathcal{A}$ first chooses two messages m_0 and m_1, which are not to be asked for the ciphertext previously, and sends them to the challenger. Then, the challenger picks a random element $b \in \{0,1\}$, and gives $\mathcal{A}$ the ciphertext $C_{m_b}=EMBEnc(U_{pub},S_{pub},m_b)$. Finally, $\mathcal{A}$ outputs a guess $b' \in \{0,1\}$ for b. We define the advantage of $\mathcal{A}$ in breaking the SPKS scheme as $Adv_{\mathcal{A}}(k)=|Pr[b=b']-\frac{1}{2}|$. We say that the SPKS scheme is semantically secure if for any polynomial time adversary $\mathcal{A}$, the function $Adv_{\mathcal{A}}(k)$ is negligible.

5. Construction of the SPKS scheme

In this section, we construct the SPKS scheme using the bilinear maps. Let $\mathcal{IG}$ be some BDH parameter generator, which runs in polynomial time to generate a prime q, two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order q, a random generator g of $\mathbb{G}_1$, and a bilinear map $\hat{e}: \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$. Let H_1, H_2, H_3, H_4 and H_5 are random oracles, where $H_1,H_3: \{0,1\}^* \to \mathbb{G}_1^*$, $H_2,H_5: \mathbb{G}_2 \to \{0,1\}^{log^q}$, and $H_4: \mathbb{G}_2 \to \{0,1\}^n$. The plaintext space includes $M \in \{0,1\}^n$ for some n and $W \in \{0,1\}^*$. The ciphertext space includes $C_M=\mathbb{G}_1^* \times \{0,1\}^n$ and $C_W \in \mathbb{G}_2$. We can either pad the shorter emails or split the longer emails to make all the emails an equal length n. We present our scheme by describing the following seven algorithms:

1. *KeyGen*: Given a sufficiently large security parameter $K_1 \in \mathbb{Z}^+$, it picks a random element $x \in \mathbb{Z}_q$ and computes g^x. A user's public key is g^x with the corresponding private key x. Given a sufficiently large security parameter $K_2 \in \mathbb{Z}^+$, it picks a random element $y \in \mathbb{Z}_q$ and computes g^y. In addition, the CSP's public key is g^y with the corresponding private key y.
2. *EMBEnc*: To encrypt an email m under a user's public key g^x and the CSP's public key g^y, it picks a random element $r \in \mathbb{Z}_q$ and a random element $\rho \in \{0,1\}^{log^q}$, computes $u_1=g^r$, $u_2=\rho \oplus H_5(\hat{e}(g^x,g^y)^r)$, and $u_3=m \oplus H_4(\hat{e}(H_3(\rho),(g^x)^r))$, and sets the ciphertext $C_m=(u_1,u_2,u_3)$.
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 $H_2(\hat{e}(g^x, H_1(W_i)^r))$, where $W_i \in \{W_1,\ldots,W_k\}$, sets the ciphertext $C_{W_i}=H_2(\hat{e}(g^x, H_1(W_i)^r))$, and sends the following message to the CSP:

   $MSG'_{U2CSP}=[C_m,C_{W_1},\ldots,C_{W_k}]$.

4. *TCompute*: To retrieve only the 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 CSP.
5. *KWTest*: To determine whether a given email contains keyword W_j, it tests whether $C_{W_i}=H_2(\hat{e}(u_1, T_{W_j}))$. If so, $KWTest(U_{pub},C_{W_i}, T_{W_j})$ outputs 1. Otherwise, it outputs 0. Note that if $W_i=W_j$, then $C_{W_i}=H_2(\hat{e}(g^x,H_1(W_i)^r))= H_2(\hat{e}(g^r, H_1(W_j)^x))=H_2(\hat{e}(u_1,T_{W_j}))$ as required.
6. *PDecrypt*: To get an intermediate result of the decipherment, it calculates ρ, computes $C_\rho=\hat{e}(H_3(\rho),u_1)$, and sends the following results to the user:

   $MSG_{CSP2U}=[C_m,C_{W_1},\ldots,C_{W_k},C_\rho]$.

   Note that $\rho=u_2 \oplus H_5(\hat{e}(g^x,g^y)^r) = u_2 \oplus H_5(\hat{e}(g^x,g^r)^y)$. Therefore, it could calculate ρ using the CSP's private key y.
7. *Recovery*: Given the ciphertext $C_m=(u_1,u_2,u_3)$ and C_ρ, it computes $m=u_3 \oplus H_4((C_\rho)^x)$ to recover the message m. Note that: $m=u_3 \oplus H_4(\hat{e}(H_3(\rho),(g^x)^r))=u_3 \oplus H_4(\hat{e}(H_3(\rho),g^r)^x)= u_3 \oplus H_4((C_\rho)^x)$.

For the sake of reducing the computational overhead and increasing the searching speed, the CSP could calculate ρ as soon as it receives MSG'_{U2CSP}, and stores the message as follows:

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

Security intuition: We provide the security intuition of the SPKS scheme, whose security will be proved in Appendix A. According to the SPKS scheme, the CSP is able to calculate ρ using its private key y. Suppose the CSP 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 cannot calculate $\hat{e}(g,g)^{arx}$, assuming that the BDH problem is hard. In other words, the CSP needs to compute $m=u_3 \oplus H_4(\hat{e}(H_3(\rho),(g^x)^r))=u_3 \oplus H_4(\hat{e}(g,g)^{arx})$ to recover the plaintext, which corresponds to computing the BDH problem. Therefore, the CSP, seeing only a random value ρ and calculating an intermediate result of the decipherment, has no idea about what the plaintext is.

6. Performance analysis

The main difference between PEKS and SPKS is that the former uses the standard public key encryption algorithm to encrypt an email without specifying its specific implementation, requiring all the decryption to be done by a user, whereas the latter uses the *EMBEnc* algorithm to encrypt an email, which encrypts an email under a user's public key and the CSP's public key to enable the CSP to participate in the partial decipherment. We claim that SPKS enables the CSP to participate in the decipherment to reduce the computational cost of a thin client, and thus it is more adaptable to a cloud environment than PEKS. To validate our claim, we compare the performance of SPKS with that of PEKS during encrypting and decrypting an email in a user's view in this section.

Please cite this article as: Liu Q, et al. Secure and privacy preserving keyword searching for cloud storage services. J Network Comput Appl (2011), doi:10.1016/j.jnca.2011.03.010

Here, we first assume some preconditions as follows:

1. Let q be a prime satisfying $p=2$ *mod* 3 and let $q>3$ be a prime factor of $p+1$. Let E be the elliptic curve defined by the equation $y^2=x^3+1$ over $\mathbb{F}_p$. We use the Weil pairing on elliptic curve E to implement SPKS, and use the ElGamal encryption based on Elliptic Curve Cryptography (ECC) over $\mathbb{F}_p$ to implement PEKS. Here, the group $\mathbb{G}_1$ is the subgroup of order q of the group of points on the elliptic curve $y^2=x^3+1$ over $\mathbb{F}_p$, the group $\mathbb{G}_2$ is the subgroup of order q of $\mathbb{F}_{p^2}^*$, and the bilinear map $e:\mathbb{G}_1\times\mathbb{G}_1\to\mathbb{G}_2$ is the Weil pairing on the elliptic curve $E(\mathbb{F}_p)$.
2. We use L, $\{0,1\}^n$, $\{0,1\}^n$ and k to denote the length of an email, the plaintext space of SPKS, the plaintext space of PEKS, and the number of keywords in an email, respectively.

For the purpose of discussion, we simply introduce the ElGamal encryption based on Elliptic Curve Cryptography (ECC) over $\mathbb{F}_p$.

1. *KeyGen*: It picks a random element $x\in\mathbb{F}_p$, and computes g^x. The user's public key is g^x with the corresponding private key x.
2. *Encryption*: To encrypt an email m, it translates m into an element in $\mathbb{F}_p$, picks a random element $r\in\mathbb{F}_p$, computes $u_1=g^r$, $u_2=m\cdot(g^x)^r$, and sets ciphertext $C_m=(u_1,u_2)$.
3. *Decryption*: Given the ciphertext $C_m=(u_1,u_2)$, it computes $m=u_2\cdot((u_1)^x)^{-1}$ to recover the message m.

Next, we compare the performance of SPKS with that of PEKS under the following two cases:

Case I: n is sufficiently large such that it is either larger than or the same as the maximal length L of any email. In this case, we pad the shorter message to make all the emails to have the equal length n.

(1) Computational cost of encryption: The *EMBEnc* algorithm requires to first compute the point multiplication on the curve $E(\mathbb{F}_p)$ once to get $u_1=g^r$, then compute the Weil pairing of g^x and g^y once, the modular exponentiation on $\mathbb{F}_{p^2}^*$ once, the hash function from $\mathbb{F}_{p^2}^*$ to $\{0,1\}^{\log^q}$ once, and the XOR operation once to get $u_2=\rho\oplus H_5(e(g^x,g^y)^r)$, finally compute the hash function from $\{0,1\}^*$ to a point of the curve $E(\mathbb{F}_p)$ once, the point multiplication on the curve $E(\mathbb{F}_p)$ once, the Weil pairing of $H_3(\rho)$ and $(g^x)^r$ once, the hash function from $\mathbb{F}_{p^2}^*$ to $\{0,1\}^n$ once, and the XOR operation once to get $u_3=m\oplus H_4(e(H_3(\rho),(g^x)^r))$. The computation for the Weil pairing of g^x and g^y is independent of the message to be encrypted, and hence can be done once for all. Therefore, SPKS generally needs to execute the Weil pairing operation once, the point multiplication operation twice, the modular exponentiation once, hash function operation three times, and XOR operation twice, whereas PEKS needs to execute the point multiplication operation twice and the modular multiplication once to encrypt an email.

The detailed comparison results are given in Table 1. We use *map*, *mul*, *exp*, *mod*, *has*, and *xor* as abbreviations for the executions of the Weil pairing operation, the point multiplication operation, the modular exponentiation operation, the modular multiplication operation, the hash function operation, and the XOR operation, respectively.

Several studies (Zhu et al., 2005) have shown that the most expensive computation is the computation for the Weil pairing, the next is the point multiplication and the modular exponentiation, the third is the operation on the finite field, such as the modular multiplication, and the least is the hash function and the XOR operation. From Table 1, we know that SPKS has more computational cost than PEKS, due to the fact that SPKS needs to execute the Weil pairing once to encrypt an email.

(2) Communication cost of encryption: The *EMBEnc* algorithm requires to send the additional log^q-bits length ciphertext u_2 to the CSP, and thus SPKS has a little more communication cost than PEKS.

(3) Computational cost of decryption: The *Recovery* algorithm requires to first compute the modular exponentiation on $\mathbb{F}_{p^2}^*$ once, then compute the hash function from $\mathbb{F}_{p^2}^*$ to $\{0,1\}^n$ once, finally execute the XOR operation once to decrypt the ciphertext. The computation for one bilinear map, which is the most expensive operation, has been completed by the CSP. Therefore, SPKS generally needs to execute the modular exponentiation once, the hash function operation once, and the XOR operation once, whereas PEKS needs to execute the point multiplication operation once, the inversion operation once, and the modular multiplication once to decrypt an email.

The detailed comparison results are given in Table 2. We use the same symbols as those in Table 1, and use *inv* as the abbreviation for the execution of the inversion operation.

From Table 2, we know that the computational cost of decryption in SPKS is less than that in PEKS.

(4) Communication cost of decryption: The *Recovery* algorithm in SPKS requires to receive C_ρ together with the ciphertext of an email, which could be used to recover the plaintext with lower computational cost. The CSP needs to send the additional log^q-bits C_ρ to a user, and thus SPKS has a little more communication cost than PEKS.

Next, we also show that SPKS has better performance than PEKS in Case II.

Case II: n is a relatively small number in comparison with the average length of most emails. In this case, we need to split a longer email into several segments and pad the last segment to make each segment to have the equal length n.

(1) Computational cost of encryption: We encrypt each segment of an email with the same parameters r and ρ, therefore, the computation for $u_1=g^r$ and $u_2=\rho\oplus H_4(\hat{e}(g^x,g^y)^r)$, and the partial computation for u_3 that is $H_4(\hat{e}(H_3(\rho),(g^x)^r))$ only needs to be done once for an email. If an email needs to be split into M segments, then SPKS generally needs to execute the Weil pairing operation once, the point multiplication operation twice, the modular exponentiation once, the hash function operation three times, and XOR operation $M+1$ times, whereas PEKS needs to execute point multiplication operation twice and the modular multiplication

Table 1
Comparison of computational cost of encryption in Case I.

Operation	PEKS	SPKS
map	0	1
mul	2	2
exp	0	1
mod	1	0
has	0	3
xor	0	2

Table 2
Comparison of computational cost in decryption in Case I.

Operation	PEKS	SPKS
map	0	0
mul	1	0
exp	0	1
inv	1	0
mod	1	0
has	0	1
xor	0	1

Please cite this article as: Liu Q, et al. Secure and privacy preserving keyword searching for cloud storage services. J Network Comput Appl (2011), doi:10.1016/j.jnca.2011.03.010

M times to encrypt the M segments of an email under the same conditions.

The detailed comparison results are given in Table 3. We use the same symbols as those in Table 1, and assume that an email needs to be split into M segments.

From Table 3, we know that as the number of segments of an email increases, there is only a small computational cost increase for executing one XOR operation in SPKS, whereas a large computational cost increase for executing one modular multiplication operation in PEKS to encrypt the email.

(2) Communication cost of encryption: We encrypt each segment of an email with the same parameters r and ρ, but only append to the first segment with the k encrypted keywords. As described in Section 4, for the sake of reducing the computational overhead and increasing the searching speed, the CSP could calculate ρ as soon as it receives MSG'_{U2CSP}, and stores the message as follows:

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

Therefore, the CSP is able to know which segments belong to the same email by examining the value of ρ, provided that we choose different ρ for each email. To determine whether a given email contains keyword W_j, the CSP executes the *KWTest* algorithm once, and returns all the segments of the email. We need to append to each segment of an email with its k encrypted keywords in PEKS, because the CSP cannot know which segments belong to the same email. If an email should be split into M segments, then SPKS only needs to send the keywords once for an email, but PEKS needs to send the keywords M times. In other words, as the number of segments of an email increases, there is only a small communication cost increase for sending the additional log^q-bits ciphertext u_2 in SPKS, whereas a large communication cost increase for sending the additional $k \cdot log^q$-bits keywords in PEKS to encrypt the email.

(3) Computational cost of decryption: For the same reason, as the number of segments of an email increases, there is only a small computational cost increase for executing one XOR operation in SPKS, whereas more computational cost increases for executing one modular multiplication operation in PEKS to decrypt the email.

The detailed comparison results are given in Table 4. We use the same symbols as those in Table 2, and assume that an email needs to be split into M segments.

Table 3
Comparison of computational cost of encryption in Case II.

Operation	PEKS	SPKS
map	0	1
mul	2	2
exp	0	1
mod	M	0
has	0	3
xor	0	$M+1$

Table 4
Comparison of computational cost of decryption in Case II.

Operation	PEKS	SPKS
map	0	0
mul	1	0
exp	0	1
inv	1	0
mod	M	0
has	0	1
xor	0	M

(4) Communication cost of decryption: The communication cost of decryption in Case II is the same as that in Case I.

Therefore, the encryption cost of SPKS is comparable to that of PEKS, but the decryption cost of SPKS is much less than that of PEKS in the case of large emails.

7. Conclusion

In this paper, we investigated the characteristics of cloud storage and proposed the SPKS scheme for cloud storage services. It allows the CSP to participate in the decipherment, thus a user could pay less computational overhead for decryption. Furthermore, it is a searchable encryption scheme, thus the CSP could search the encrypted files efficiently without leaking any information. It is provable that the proposed scheme has semantic security against adaptive chosen plaintext attacks. By performance analysis, we show that our scheme outperforms the scheme proposed by Boneh et al. (2004) when applied to a cloud environment.

Acknowledgements

This work is supported by the National Natural Science Foundation of China under Grant Nos. 61073037 and 90718034, Hunan Provincial Science and Technology Program under Grant Nos. 2010GK2003 and 2010GK3005, and Changsha Science and Technology Program under Grant Nos. K1003064-11 and K1003062-11.

Appendix A. Security proof

To analyze the security of the SPKS scheme, we provide the following theorem, which shows that the SPKS scheme is semantically secure under the BDH assumption:

Theorem A.1. *Let H_1 and H_2 be random oracles from $\{0,1\}^*$ to $\mathbb{G}_1^*$ and from $\mathbb{G}$ to $\{0,1\}^n$, respectively. Suppose $\mathcal{A}$ is an IND-CPA adversary that has the advantage ε against the SPKS scheme. Suppose $\mathcal{A}$ makes $q_{H_2} > 0$ hash function queries to H_2. Then, there is an algorithm $\mathcal{B}$ that solves the BDH problem with the advantage at least $\varepsilon' = 2\varepsilon/q_{H_2}$ and a running time $O(time(\mathcal{A}))$.*

Proof. $\mathcal{B}$ is given $\rho \in \{0,1\}^{log^q}$, $\mu_0 = g$, $\mu_1 = g^\alpha$, $\mu = g^\beta$, $\mu_1 = g^\gamma \in \mathbb{G}_1$, where α, β, γ are random elements in $\mathbb{Z}_q$. Its goal is to output $D = \hat{e}(g,g)^{\alpha\beta\gamma} \in \mathbb{G}$. Let D be the solution to the BDH problem. $\mathcal{B}$ finds D by interacting with $\mathcal{A}$ as follows:

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

H_1-queries: $\mathcal{B}$ maintains a list of tuples called H_1-list, in which each entry is a tuple of the form $\langle \rho_j, f_j \rangle$. The list is initially empty. When $\mathcal{A}$ queries H_1 at a point of ρ_i, $\mathcal{B}$ checks if $\rho_i = \rho_j$ where ρ_j already appears on H_1-list in the form of $\langle \rho_j, f_j \rangle$. If so, $\mathcal{B}$ responds to $\mathcal{A}$ with $H_1(\rho_i) = f_j$. Otherwise, $\mathcal{B}$ picks a random element $d \in \mathbb{Z}_q$, computes $f_i = \mu \cdot g^d = g^\beta \cdot g^d \in \mathbb{G}_1^*$, adds the tuple $\langle \rho_i, f_i \rangle$ to H_1-list, and responds to $\mathcal{A}$ with $H_1(\rho_i) = f_i$.

H_2-queries: $\mathcal{B}$ maintains a list of tuples called H_2-list, in which each entry is a tuple of the form $\langle r_j, l_j \rangle$. The list is initially empty. When $\mathcal{A}$ queries H_2 at a point of r_i, $\mathcal{B}$ checks if $r_i = r_j$ where r_j already appears on H_2-list in the form of $\langle r_j, l_j \rangle$. If so, $\mathcal{B}$ responds to $\mathcal{A}$ with $H_2(r_i) = l_j$. Otherwise, $\mathcal{B}$ picks a random string $l_i \in \{0,1\}^n$, adds the tuple $\langle r_i, l_i \rangle$ to H_2-list, and responds to $\mathcal{A}$ with $H_2(r_i) = l_i$.

Challenge: $\mathcal{A}$ outputs two messages m_0 and m_1 on which it wishes to be challenged. $\mathcal{B}$ randomly picks $b \in \{0,1\}$ and a random string $S \in \{0,1\}^n$, and gives the ciphertext $C = (\mu_1, S)$ to $\mathcal{A}$.

Please cite this article as: Liu Q, et al. Secure and privacy preserving keyword searching for cloud storage services. J Network Comput Appl (2011), doi:10.1016/j.jnca.2011.03.010

The decryption of the ciphertext is

$$m_b = S \oplus H_2(\hat{e}(H_1(\rho),\mu_1)^{\gamma}) = S \oplus H_2(\hat{e}(H_1(\rho),g^{\alpha})^{\gamma})$$
$$= S \oplus H_2(\hat{e}(g^{\beta}\cdot g^{d},g^{\alpha})^{\gamma}) = S \oplus H_2(\hat{e}(g,g)^{\alpha\gamma(\beta+d)}).$$

Hence, C is a valid ciphertext for m_b as required.

Guess: $\mathcal{A}$ outputs its guess $b' \in \{0,1\}$ for b. $\mathcal{B}$ picks a random pair $\langle r_i, l_i \rangle$ from H_2-list and outputs r_i as the solution to the given instance of BDH.

To complete the proof of Theorem A.1, we now show that $\mathcal{B}$ correctly outputs D with the probability at least $2\varepsilon/q_{H_2}$. Let Q be the event that $\mathcal{A}$ issues a query for ν. If $\neg Q$, we know that the decryption of the ciphertext is independent of $\mathcal{A}$'s view. Let $Pr[b=b']$ be the probability that $\mathcal{A}$ outputs the correct result, therefore, in the real attack $Pr[b=b'|\neg Q]=\frac{1}{2}$. Since $\mathcal{A}$ has the advantage ε, $|Pr[b=b'|\neg Q]-\frac{1}{2}|\geq\varepsilon$. According to the following formulae, we know $Pr[Q]\geq 2\varepsilon$:

$$Pr[b=b'] = Pr[b=b'|\neg Q]Pr[\neg Q]+Pr[b=b'|Q]Pr[Q]$$
$$\leq \tfrac{1}{2}Pr[\neg Q]+Pr[Q] = \tfrac{1}{2}+\tfrac{1}{2}Pr[Q],$$

$$Pr[b=b'] \geq Pr[b=b'|\neg Q]Pr[\neg Q] = \tfrac{1}{2}Pr[\neg Q] = \tfrac{1}{2}-\tfrac{1}{2}Pr[Q].$$

Therefore, we have $Pr[Q]\geq 2\varepsilon$ in the real attack. That is to say, $\mathcal{A}$ will issue a query for l with the probability at least 2ε. $\mathcal{B}$ will choose the correct pair with the probability at least $1/q_{H_2}$ and thus, $\mathcal{B}$ produces the correct answer with the probability at least $\varepsilon' = 2\varepsilon/q_{H_2}$ as required.

References

Boneh D, Crescenzo G, Ostrovsky R, Persiano G. Public key encryption with keyword search. In: Proceedings of Eurocrypt 2004, Lecture notes in computer science, vol. 3027; 2004. p. 506–22.

Boneh D, Franklin M. Identity based encryption from the weil pairing. SIAM Journal of Computing 2003;32(3):586–615 [also as an extended abstract in Crypto, 2001].

Boneh D, Waters B. Conjunctive, subset, and range queries on encrypted data. In: Proceedings of TCC 2007, Lecture notes in computer science, vol. 4392; 2007. p. 535–54.

Chang Y.-C, Mitzenmacher M. Privacy preserving keyword searches on remote encrypted data. In: Proceedings of ACSN 2005, Lecture notes in computer science, vol. 3531; 2005. p. 442–55.

Diament T, Lee HK, Keromytis AD, Yung M. The dual receiver cryptosystem and its applications. In: Proceedings of the ACM CCS; 2004. p. 330–43.

Haclgiimfi H, Iyer B, Li C, Mehrotra S. 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.

Liu Q, Wang G, Wu J. An efficient privacy preserving keyword search scheme in cloud computing. In: Proceedings of IEEE TrustCom-09 in conjunction with IEEE CSE-09; 2009. p. 715–20.

Shi E, Bethencourt J, Chan T-HH, Song D, Perrig A. Multi-dimensional range query over encrypted data. In: Proceedings of IEEE symposium on security and privacy; 2007. p. 350–64.

Song DX, Wagner D, Perrig A. Practical techniques for searches on encrypted data. In: Proceedings of the 2000 IEEE symposium on security and privacy; 2000. p. 44–55.

Weiss A. Computing in the clouds. netWorker 2007;11(4):16–25.

Zhu R, Yang G, Wong D. An efficient identity-based key exchange protocol with KGS forward secrecy for low-power devices. In: Internet and network economics: Proceedings of the first international workshop WINE, Lecture notes in computer science, vol. 3828; 2005. p. 500–09.

Please cite this article as: Liu Q, et al. Secure and privacy preserving keyword searching for cloud storage services. J Network Comput Appl (2011), doi:10.1016/j.jnca.2011.03.010