Improvement of Multi-user Searchable Encrypted Data Scheme

Tran Thao Phuong, Kazumasa Omote
Japan Advanced Institute of Science and Technology
1-1 Asahidai, Nomi, Ishikawa, Japan
Email: {tpthao, omote}@jaist.ac.jp

Nguyen Gia Luyen, Nguyen Dinh Thuc
University of Science HCMC
227 Nguyen Van Cu, HCMC, Vietnam
Email: gialuyen1990@gmail.com, ndthuc@fit.hcmus.edu.vn

***Abstract*—Nowadays, since the amount of information is increasing quickly, data owners have tendency to publish their data to external service provider called Cloud Computing. Using Cloud Computing, users can reduce the burden of data management. However, since the service provider is not fully trusted, there are many challenges in securing the data stored on Cloud: integrity, availability and privacy. In this paper, we focus mainly on the third one, privacy.**

To ensure privacy of outsourced data, the common solution is to encrypt the data so that the server cannot know the plaintext. The problem is that: how can the authorized users access the encrypted data and how can the server perform its allowed operations for user's request, i.e, searching, modifying, inserting, deleting such encrypted data without knowing the plaintext.

To address this problem, researchers have proposed *multi-user searchable encrypted data schemes* which supports encrypted queries over encrypted data. Especially, C. Dong et al [1] are the newest recent authors proposing a new *multi-user searchable encrypted data scheme* which does not need to depend on shared key among multiple users and is efficient in key revocation. Unfortunately, C. Dong's scheme cannot thoroughly deal with a dangerous and common attack in Cloud: *collusion attack* which happens when a certain user colludes with the server to learn the secret information.

In this paper, we improve C. Dong's scheme to become a completely secure scheme. Firstly, our scheme can efficiently and thoroughly prevent collusion attack with low overhead by using only one server. Secondly, to satisfy the strictly condition: the server is untrusted, our scheme does not allow the server to keep any secrete information, unlike C. Dong's scheme in which the server keeps a partial secrete information.

***Index Terms*—cloud computing, multi-user searchable encrypted data schemes, collusion attack**

I. Introduction

Nowadays, many individuals and organizations outsource their data to remote cloud service providers called Cloud Computing. Such outsourcing of data enables customers to store more data on external storage than on private computer systems. Also, it permits the users to manage their data easily because the users can share and access their data from anywhere through a Web.

In Cloud Computing, the researchers have considered many scenarios. In this paper, we consider the scenario in which there are multiple users who publish their data to the server hosted by third party and can access the data, called *authorized users*. If a certain user is no longer allowed to access the shared data, he will be revoked from the system that we call *revoked user*. The data stored on the server is shared to all authorized users so that they can use the same data storage in the server. In this model, only the authorized users are fully trusted, the revoked users and the server are untrusted. Furthermore, the data stored within the server is treated as dynamic data. Users are not only able to read the data but also can do the update operations such as modification, insertion, deletion.

Although the method of outsourcing reduces storage burden for users, the problem is: the server is typically untrusted. Thus, this model introduces numerous interesting research challenges: (i) data privacy, (ii) data availability, (iii) data integrity.

- *Data privacy*: The data needs to be prevented from the disclosure of the outsourced data to unauthorized individuals or systems. The system attempts to enforce privacy by encrypting the data or by restricting access.
- *Data availability*: For any information system to serve its purpose, the data must be available when it is needed. This means the computing systems used to store and process the information, the security controls used to protect it, and the communication channels used to access it must be functioning correctly. High availability systems aim to remain available at all times, preventing service disruptions due to power outages, hardware failures, and system upgrades.
- *Data integrity*: Integrity means the data cannot be modified undetectably. Integrity is violated when the data is actively modified.

In this work, we focus on the first issue, data privacy. To ensure privacy of outsourced data, the common approach is that users encrypt the data before sending encrypted data to the server. The problem is: if a user wants to retrieve the data stored in the server, how can the server perform data searching on encrypted data to find the corresponding results for such user. To address this problem, some *multi-user searchable encrypted data schemes* have been proposed.

In the scenario that a group of users share data through an untrusted server, researchers introduces many approaches based on *multi-user searchable encrypted data* scheme. Song et al [6] are the first authors introducing their approach in which user uses keyed hash for keywords so that the server can search on encrypted data without knowing plaintext. Goh

[7] then presented secure index based on bloom filter technique that is a space-efficient probabilistic data structure used to test whether an element is a member of a set. Thus, Goh's scheme can reduce storage cost in searchable encrypted data scheme. D. Boneh et al. [3] introduced public key for keyword search in which users can use public key to encrypt data and send encrypted data to the server. Only authorized users who have private key can search and decrypt the data. R. Curtmola [5] proposed the scheme based on broadcast encryption and not use asymmetric encryption like Boneh's scheme.

However, most of the above schemes exist important weak points. Firstly, the schemes depend on a single set of secret keys of users or shared keys among multiple users. Therefore, key revocation requires high cost since the data needs to be re-encrypted by the new key when revocation occurs. Secondly, shared data is only readable, thus users cannot do update operations, i.e., modification, insertion, deletion.

C. Dong et al [1] are the recent authors proposing a new multi-user searchable encrypted data scheme to overcome those weak points:

- This scheme does not need shared key between the groups of authorized users, instead each authorized user has his own key. When a user no longer has permission of access to the data, revoking key of this user neither requires data re-encryption nor affects keys of the remaining users. Thus, the main achievement of their scheme is to minimize computation and communication cost in key revocation.
- This scheme supports dynamic data. In previous works, the whole document set is encrypted before searching can take place, afterwards no modification is allowed on the document set. Meanwhile, in this scheme, the document set can be continuously updated by authorized users.

Nevertheless, this scheme encounters a difficulty that it cannot resist a serious and common attack in a multi-user searchable encrypted data scheme: *collusion attack*. Collusion attack is dangerous because it happens when a certain user colludes with the server to obtain the secret information, i.e, the master key. Once they know the master key, they can learn the keys of all users in the system and can decrypt all the encrypted data. Unless this attack is strictly prevented, the multi-user searchable encrypted data scheme becomes meaningless. Actually, C. Dong et al. discussed their solution for resisting collusion attack: employ multiple servers, split the master key into multiple shares and introduce to these multiple servers from storage server providers competing with each other.

However, the question is: how can we ensure whether the storage servers collude with each other or not? because in reality, everything can happen with untrusted service providers. Thus, their solution which splits the master key to multiple share and distribute to different servers is not practical.

Furthermore, besides the security aspect, we consider the efficiency aspect. A scheme which needs multiple servers is inefficient since the model requires more cost to employ more servers.

Contributions Motived from the good ideas of C. Dong about multi-user searchable encrypted data scheme, we improve this scheme to become a completely secure and efficient scheme.

- Firstly, our scheme thoroughly protects against the dangerous attack: *collusion attack* since we ensure that the server cannot collude with other entities, and the server cannot forge a user in other to learn about the secret information.
- Secondly, satisfying the strictly condition in Cloud: the server is untrusted, our scheme does not give any secrete information for the server even just a partial information, unlike C. Dong's scheme.
- Thirdly, we only need one server instead of multiple servers, unlike C. Dong's solution.

Organization In Section 2, we describe preliminaries. We review previous work in Section 3. Our proposed scheme are described in Section 4, and its security and efficiency analysis in Section 5. We state conclusion and future work in Section 6.

II. Preliminaries

A. System model

Based on the scenario that there is a group of users sharing data through a server, we identify three types of entities:

- Server: This entity is the data storage hosted by a third party, and is untrusted.
- Users: There are two kinds of users in the system:
  - *Authorized user*: This user is fully trusted, and can search, update the encrypted data on the server-side.
  - *Revoked users*: If a certain authorized user is withdrawn his privilege and can no longer access the data stored in the server, this entity is called revoked user and is untrusted.
- TS (Trusted server): This entity is fully trusted server which has two main tasks: The first task is to generate and manage the keys for all users and the server in the system. The second task is to perform a partial searching for user. We will describe more details later.

The users communicate with the server through a web server which helps to deliver the data that can be accessed through the Internet by using a Web browser.

B. Adversarial model

In our scheme, we treat the server and revoked user as adversaries. We model the server as honest but curious. The server has full access to the encrypted data stored on it and also observe the communication to and from users. Otherwise, the revoked user is excluded from the system, he does not have any privilege on the system. Moreover, we consider a dangerous attack usually happening in a multi-user searchable encrypted data scheme: collusion attack.

Collusion attack: This attack occurs when the server colludes with an certain authorized user. More concretely, if a certain authorized user gives his key to the server, the server will combine the server's key and the user's key to calculate the master key. Once they know the master key as secrete information, they can learn the keys of all remaining users and also can decrypt all encrypted data.

In another case, the server can forge an authorized user in the system, as a result he has both the key in client-side and server-side to obtain the master key easily and then can decrypted all data.

C. Unidirectional ElGamal encryption scheme

We will use this unidirectional ElGamal scheme [2] to allow multiple users to encrypt and decrypt data using private keys. Firstly, the ElGamal encryption consists of three components: $\varepsilon = (KeyGen, Enc, Dec)$.

- $KeyGen(1^k)$: On input the security parameter k, this algorithm outputs the public key $EK = (g, p, q, y)$ and the secret key $DK = (x)$, where p is a prime number, g is a generator for the $\mathbb{Z}_{p^*}$, x is randomly chosen from $\mathbb{Z}_q$, and $y = g^x \bmod p$.
- $Enc(DK, m)$: The encryption algorithm is defined as $e = Enc_{EK}(m) = (g^r \bmod p, mg^{xr} \bmod p)$, where r is chosen at random from $\mathbb{Z}_q$
- $Dec(DK, e)$: The decryption algorithm computes the message m from e by dividing mg^{xr} to $(g^r)^x \bmod p$.

The unidirectional ElGamal encryption scheme is defined as $\varepsilon = (UniGen, UniEnc, UniDec, PDec, FDec)$. For each user U:

- $UniGen$: The key generation algorithm generates a public-key pair (EK, DK) and splits the secret key $DK = x$ into two parts x_1 and x_2 such that $x = x_1 + x_2$. The proxy P receives $DK_P = x_1$ and the user F receives $DK_F = x_2$.
- $UniEnc(DK, m), UniDec(DK, e)$: same as Enc and Dec of the standard ElGamal scheme.
- $PDec, FDec$: same as Dec under x_1, respectively x_2.

The unidirectional encryption scheme is correct because $FDec_{x_2}(PDec_{x_1}(Enc_y(m))) = FDec_{x_2}(mg^{xr}/(g^r)^{x_1}) = mg^{x_2 r}/(g^r)^{x_2} = m$.

D. Public key encryption with keyword search (PEKS)

We will use this scheme in other to support searching on the encrypted data by encrypted keywords. A non-interactive public key encryption with keyword search scheme [3] consists of the following polynomial time randomized algorithms:

- $KeyGen(s)$: Takes a security parameter s, this algorithm generates a public/private key pair A_{pub} and A_{priv}
- $PEKS(A_{pub}, W)$: For a public key A_{pub} and a word W, this algorithm produces a searchable encryption of W.
- $Trapdoor(A_{priv}, W)$: Given a private key A_{priv} and a word W produces a trapdoor T_W.
- $Test(A_{pub}, S, T_W)$: Given a public key A_{pub}, a searchable encryption $S = PEKS(A_{pub}, W_0)$, and a trapdoor $T_W = Trapdoor(A_{priv}, W)$, this algorithm outputs *yes* if $W = W_0$ and *no* otherwise.

III. Recent work - C. Dong's scheme

As we mentioned in Section I, this scheme has the main achievements:

- The server can search on the encrypted data using the encrypted keywords.
- The scheme does not depend on shared keys between the groups of users, instead each authorized user has his own key. It is easy to revoke key without affecting other users and without re-encrypting the data on server-side.
- The users in the group can both read and write on the shared data.

A. The model

The scheme also considers three types of entities like our system model, but there is a difference:

- Users: same as *users* entity in our model.
- Server: same as *server* entity in our model.
- KMS (Key management server): this entity is also untrusted server, but the difference from *TS (trusted server)* entity in our model is that: KMS is only responsible for managing keys while TS in our model is responsible for managing keys and generating the trapdoor for searching which we will discuss TS later in our scheme.

B. The scheme

This scheme has eight algorithms as follows:

1) $Init(1^k)$:
KMS performs this algorithm to make the public and private information.
   - On input the security parameter 1^k, output a group G of the prime order q with a generator g.
   - Choose $x \overset{rand}{\leftarrow} \mathbb{Z}_q^*$ to compute $h = g^x$.
   - Choose a hash function H such that $\{0,1\}^* \rightarrow G$ and a pseudo-random function f with a random seed s for that f.
   - Publicize (G, g, q, h, H, f) and keep private $MSK = (x, s)$ as master key.
2) $Keygen(MSK, i)$:
KMS performs this algorithm to generate the master key and share a part of private key for the user i and the server.
   - For each user i where i is the user index, KMS chooses $x_{i_1} \overset{rand}{\leftarrow} \mathbb{Z}_q^*$ to compute $x_{i_2} = x - x_{i_1}$.
   - Securely transmit $K_{u_i} = (x_{i_1}, s)$ to user i and $K_{s_i} = (i, x_{i_2})$ to the server.
   - After receiving K_{s_i} from KMS, the server inserts K_{s_i} into its user-key mapping set $K_s = K_s \cup K_{s_i}$.
3) $Enc(K_{u_i}, D, Kw(D))$:
User partially encrypts data using his private key.
   - The user i uses his key K_{u_i} to compute ciphertext $c_i^*(D) = (g^r, g^{r x_{i_1}} D)$ of the document D where $r \overset{rand}{\leftarrow} \mathbb{Z}_q^*$

- For each keyword $kw \in Kw(D)$ where $Kw(D)$ is the set of keywords of the document D, the user i encrypts kw as $c_i^*(kw) = (\hat{c_1}, \hat{c_2}, \hat{c_3})$ such that $\hat{c_1} = g^{R+\delta}$ where $\delta = f_s(kw)$, $\hat{c_2} = \hat{c_1}^{x_{i_1}}$, $\hat{c_3} = H(h^R)$ where $R \overset{rand}{\leftarrow} \mathbb{Z}_q^*$.
- User i sends $c_i^*(C, Kw(D)) = (c_i^*(D), c_i^*(kw_1), ..., c_i^*(kw_k))$ where $k = |Kw(D)|$ to the server.

4) $Re - enc(i, K_{s_i}, c_i^*(D, Kw(D)))$:
After receiving partial encrypted data from user i, the server encrypts again using its private key.
- For the ciphertext $c_i^*(D)$ received from the user i, the server uses x_{i_2} to compute $(g^r)^{x_{i_2}}.(g^{r.x_{i_1}}D) = g^{rx}D$. The ciphertext is $c(D) = (g^r, g^{rx}D)$.
- For each $c_i^*(kw) \in c_i^*(D, Kw(D))$ received from the user i, the server computes $c(kw) = (c_1, c_2)$ such that $c_1 = \hat{c_1}^{x_{i_2}} \times \hat{c_2} = h^{R+\delta}$ and $c_2 = \hat{c_3} = H(h^R)$
- The server inserts $c(D, Kw(D)) = (c(D), c(kw_1), ..., c(kw_k))$ where $k = |Kw(D)|$ into the encrypted data storage $E(D) = E(D) \cup c(D, Kw(D))$.

5) $Trapdoor(K_{u_i}, w)$:
The user i performs this algorithm to generate the trapdoor for the keyword w he wants to search.
- The user i takes his key $K_{u_i} = (x_{i_1}, s)$ and keyword w to compute trapdoor $T_i(w) = (t_1, t_2)$ where $t_1 = g^{-r}.g^{\delta'}$ and $t_2 = h^r.g^{-x_{i_1}r}.g^{x_{i_1}\delta'} = g^{x_{i_2}r}.g^{x_{i_1}\delta'}$ such that $r \overset{rand}{\leftarrow} \mathbb{Z}_q^*$, $\delta' = f_s(w)$.
- The user i sends this trapdoor to the server.

6) $Search(i, T_i(w), E(D), K_{s_i})$:
The server searches the encrypted data after getting the trapdoor from the user i.
- After receiving $T_i(w) = (t_1, t_2)$ from user i, the server computes $T = t_1^{x_{i_1}} t_2 = g^{x\delta'} = h^{\delta'}$.
- For each $c(D, Kw(D)) \in E(D)$, for each $c(kw) = (c_1, c_2) = (h^{R+\delta}, H(h^R))$, the server tests whether $c_2 = H(c_1 T^{-1})$ or not. If so, the server will return $c(D)$. However, before returning $c(D)$ to the user, the server partially decrypts $c(D)$ as $c_i'(D)$ by computing $g^{rx}D(g^r)^{-x_{i_2}} = g^{rx_{i_1}}D$. The ciphertext $c_i'(D) = (g^r, g^{rx_{i_1}}D)$ is then sent back to user i.

7) $Dec(K_{u_i}, c_i'(D))$:
User decrypts the searched result from the server to get the plaintext.
- After receiving $c_i'(D) = (g^r, g^{rx_{i_1}}D)$, the user fully decrypts $c_i'(D)$ as $g^{rx_{i_1}}D \times (g^r)^{-x_{i_1}} = D$

8) $Revoke(i)$:
When a user has no more permission to access data, server removes the key on server-side corresponding with the key of revoked user.
- Given i, the server updates user key mapping set $K_s = K_s \backslash K_{s_i}$

C. Problem statement

We can see that collusion attack easily happens in this scheme. Assume that user i gives his private key x_{i_1} to the server. After receiving x_{i_1}, the server then finds the corresponding keys on server-side x_{i_2} and compute master key by combining $x = x_{i_1} + x_{i_2}$.

To prevent collusion attack, the authors discuss a solution that: employ multiple servers, i.e, server α and server β. Server α is used to store encrypted data and server β only needs to keep its key and does not need to store data. KMS splits the master key into three parts: $x = x_{i_1} + x_{i_2\alpha} + x_{i_2\beta}$ and then sends x_{i_1} to the user i, sends $x_{i_2\alpha}$ to server α and sends $x_{i_2\beta}$ to server β. They give a strict condition that two servers α and β belong to different storage service providers competing with each other so that they must not communicate with each other.

However, the condition that two serverw α and β must compete with each other so that they cannot collude is very loose since how can we ensure that whether the servers communicate with each other or whether the server forges a user or not? In reality, server α may collude β and a certain user i' to combine their private keys. Thus, this solution is not secure and also not practical.

Furthermore, the authors's solution requires more cost since they have to employ multiple servers instead of one server.

IV. Proposed scheme

A. Main ideas

Firstly, in order to allow multiple users encrypt and decrypt data using private keys, we deploy the unidirectional ElGamal encryption scheme given in Section II-C.

Secondly, in order to support searching on the encrypted data by encrypted keywords, we use the public key encryption with keyword search (PEKS) given in Section II-D.

Thirdly, we do not share any secret information to the server, even just a partial secret information, but the server can still search on encrypted data for user's queries.

B. Scheme in details

Our scheme consists of nine algorithms:

1) $Init(1^k)$:
TS performs this algorithm to make the public and private information for the system
- Given security parameter 1^k, TS determines two groups G_1, G_2 of prime order q and a bilinear map $e : G_1 \times G_1 \to G_2$ (the construction is based on [4]), a generator g of G_1 and two hash functions $H_1 : \{0,1\}^* \to G_1$, $H_2 : G_2 \to \{0,1\}^{log_q}$
- Choose $x \overset{rand}{\leftarrow} \mathbb{Z}_q^*$, $x' \overset{rand}{\leftarrow} \mathbb{Z}_q^*$ to compute $h = g^x$, $h' = g^{x'}$
- The public key is $pk = \{G_1, G_2, g, q, h, h', H_1, H_2, e\}$ and the private key is $sk = \{x, x'\}$

2) $KeyGen(sk, i)$:
TS performs this algorithm to generate the master key

and shares a part of secrete key for user and keeps any secret information from the server.

- For each user i, TS chooses $x_{i_1} \overset{rand}{\leftarrow} \mathbb{Z}_q^*$ to compute $x_{i_2} = x - x_{i_1}$.
- x_{i_1} is securely transmitted to the user i and TS updates its user key mapping set $uk = uk \cup (i, x_{i_1}, x_{i_2})$.

3) $Enc(D, Kw(D))$:
User encrypts data, then the server stores the encrypted data from users.

- There are two things need to be encrypted: document D and its associated set of keywords $Kw(D)$.
  - $c(D) = EncDoc(D)$: Encrypt the document D by using ElGamal encryption scheme: On input D, this algorithm outputs the ciphertext $C(D) = (g^r, h^r D)$ where $r \overset{rand}{\leftarrow} \mathbb{Z}_q^*$.
  - $c(kw) = PEKS(kw)$: Encrypt the keywords of this document using PEKS: On input kw, this algorithm outputs $c(kw) = (c_1, c_2) = (g^R, H_2(t))$ where $R \overset{rand}{\leftarrow} \mathbb{Z}_q^*$ and $t = e(H_1(kw), h'^R)$
- The ciphertext of $(D, Kw(D))$ is $c(D, Kw(D)) = (c(D), c(kw_1), ..., c(kw_k))$ where $k = |Kw(D)|$. This ciphertext is sent to the server.
- The server inserts $c(D, Kw(D))$ into the encrypted data storage $E(D) = E(D) \cup c(D, Kw(D))$

4) $Pre - search(x_{i_1}, w)$:
User performs this algorithm to encrypt the keyword that he want to search.

- When the user i wants to search a keyword w, he sends his identification i and the cipher keyword $E(w) = Enc_{x_{i_1}}(r||w)$ to TS where Enc is a symmetric encryption algorithm (DES, AES...), $r \overset{rand}{\leftarrow} \{0,1\}^t$ and $||$ indicates concatenation.

5) $Trapdoor(i, E(w))$:
TS performs this algorithm to generate the trapdoor for the keyword that user i wants to search.

- After receiving $E(w)$ from the user i, TS decrypts it and discards t random bits to get w.
- TS computes $T_w = H_1(w)^{x'}$ and sends (i, T_w) to the server.

6) $Search(i, T_w)$:
The server searches the encrypted data after receiving T_w from TS.

- For each $c(D, Kw(D)) \in E(D)$, the server tests each $c(kw) = (c_1, c_2)$, if $H_2(e(T_w, c_1)) = c_2$ then returns $w = kw$ and sends $(i, c(D))$ to TS.

7) $Pre - Dec(i, c(D))$:
TS performs this algorithm to partially decrypt the searched result before sending the result for user i.

- TS decrypts $c(D)$ as $c'(D) = (g^r, h^r D \times (g^r)^{-x_{i_2}}) = (g^r, g^{r x_{i_1}} D)$
- TS sends $c'(D)$ to the user i

8) $Dec(x_{i_1}, c'(D))$:
User decrypts the result from TS to obtain the plaintext containing the keyword.

- After receiving $c'(D)$, the user i decrypts $g^{r x_{i_1}} D \times (g^r)^{-x_{i_1}} = D$

9) $Revoke(i)$:
TS revokes the key of user when that user has no more permission to access data.

- Given i, TS updates user key mapping set $uk = uk \backslash (i, x_{i_1}, x_{i_2})$

V. Discussion

A. Security analysis

1) Security of the unidirectional ElGamal encryption: The following theorem proves that the unidirectional ElGamal is as secure as the original ElGamal scheme:

Theorem 1 Let ε' = (UniGen, UniEnc, UniDec, PDec, FDec) be an unidirectional ElGamal encryption scheme. ε' is CPA secure against: the proxy P, the user F, and all users U where the adversary's success is defined as follows:

$$Succ_{P,\varepsilon'} \overset{def}{=} Pr[b = \tilde{b} | (EK, DK) \leftarrow UniGen(1^k), (m_0, m_1) \leftarrow P(EK, DK_P), b \leftarrow \{0,1\}, \tilde{b} \leftarrow P(EK, DK_P, UniEnc_{EK}(m_b))]$$

$$Succ_{F,\varepsilon'} \overset{def}{=} Pr[b = \tilde{b} | (EK, DK) \leftarrow UniGen(1^k), (m_0, m_1) \leftarrow F^{PDec*}(EK, DK_F), b \leftarrow \{0,1\}, \tilde{b} \leftarrow F^{PDec*}(EK, DK_F, UniEnc_{EK}(m_b))]$$

The proof of Theorem 1 is given in [2]

2) Security of PEKS: The security for a PEKS is in the sense of semantic-security. We need to ensure that an $PEKS(A_{pub}, W)$ does not reveal any information about W unless T_W is available. The security against an active attacker who is able to obtain trapdoors T_W for any W of his choice. Even under such attack the attacker should not be able to distinguish an encryption of a keyword W_0 from an encryption of a keyword W_1 for which he did not obtain the trapdoor. The security is defined against an active attacker A using *PEKS security game* in which the security parameter s is given to both players as input:

- The challenger runs the $KeyGen(s)$ to generate A_{pub} and A_{priv}. It gives A_{pub} to the attacker.
- The attacker adaptively asks the challenger for the trapdoor T_W for any keyword $W \in \{0,1\}^*$ of his choice.
- At some point, the attacker A sends the challenger two words W_0, W_1 on which it wishes to be challenged. The only restriction is that the attacker did not previously ask for the trapdoors T_{W_0} or T_{W_1}. The challenger picks a random $b \in \{0,1\}$ and gives the attacker $C = PEKS(A_{pub}, W_b)$. C is referred as the challenge PEKS.
- The attacker can continue to ask for trapdoors T_W for any keyword W of his choice as long as $W \neq W_0, W_1$.
- The attacker A outputs $b' \in \{0,1\}$ and wins the game if $b = b'$.

The attacker wins the game if he can correctly guess whether he was given the PEKS for W_0 or W_1. A's advantage in breaking the PEKS as:

$$Adv_A(s) = |Pr[b = b'] - \tfrac{1}{2}|$$

PEKS is semantically secure against an adaptive chosen keyword attack if for any polynomial time attacker A we have that $Adv_A(s)$ is a negligible function.

3) Collusion attack: In our scheme, after splitting the master key $x = x_{i_1} + x_{i_2}$, TS only sends x_{i_2} for user i but keeps x_{i_2} secret. The server does not have any secret information (unlike in [1] in which the server keeps x_{i_2} as his private key). Therefore, even if a user colludes with the server, they cannot learn anything about the master key by computing $x = x_{i_1} + x_{i_2}$.

We consider another case that the server does not collude with any user, instead the server forges a user by registering a user in the system. In this case, as an authorized user, the server has his own private key x_{i_1} but cannot know x_{i_2}. Thus, the master key is also protected.

B. Efficiency

1) Dynamic data: The data in our scheme is treated as dynamic data. That means users not only can search data (read) but also can do update operations (write). We describe more details all of operations user can perform:

- Search: Given a keyword, user can require the server to search on encrypted data: User executes $Pre-search()$, TS executes $Trapdoor()$, the server executes $Search()$, TS executes $Pre-Dec()$ and user executes $Dec()$ which finally returns results containing the keyword.
- Insert: Given a new document D and a set of associated keywords of this document $Kw(D)$, user first encrypt them using encryption algorithm $Enc()$, then sends encrypted data to the server for storing.
- Delete: Given a keyword, user first requires the server to search with the keyword. After finding the corresponding document for user, the server removes this document instead of sending result to user.
- Modify: User needs to perform deletion on the old document, then inserts the new document. In our scheme, user cannot modify directly on encrypted data. We leave this problem for future work.

2) Single server: As we analysed in Section of problem statement III-C, our solution does not need to employ multiple servers. In our scheme, there are 3 entities like [1]: groups of users, one trusted server (TS) and one server in Cloud.

3) Assurance that a revoked user cannot search: Considering the case that a revoked user, say u_i, tries to search based on some known keywords (He got these keywords when being an authorized user). The only thing he can do is to run $Pre - search(x_{i_1}, w)$ algorithm on input his old key x_{i_1} and keyword w. The output of this algorithm $E(w) = Enc_{x_{i_1}}(r||w)$ (where r is random and E is symmetric encryption function) is then sent to TS. After receiving $E(w) = Enc_{x_{i_1}}(r||w)$ from user u_i, TS is going to get w from $E(w) = Enc_{x_{i_1}}(r||w)$ by finding the corresponding key x_{i_1} to decrypt the symmetric encryption function E. However, it's really easy for TS to recognize that user u_i is a revoked user by figuring out that x_{i_1} is no longer in a valid user key. Even if the revoked user u_i colludes with the server, they still search nothing because processing of searching between a user and the server needs to be passed through TS.

VI. Conclusion and Future work

In this paper, we proposed a new multi-user searchable encrypted data scheme. Our scheme also allows to revoke a user easily without affecting another users and re-encrypting the data. The server can update and search on the encrypted data while learning nothing about the plaintext and the keywords. In addition, our scheme can thoroughly prevent the dangerous attack: collusion attack which commonly occurs in a multi-user searchable encrypted data scheme between a certain user and the server. Our solution is secure without requiring multi servers as well as synchronization between the servers.

However, our scheme still has two problems that we leave to future works. Firstly, to modify the data, the user needs to perform deletion on the old document, then inserts the new document. In our scheme, we cannot modify directly on encrypted data. We leave this problem for future work. Secondly, we try to think out a protocol which does not need the TS (trusted server) any more, instead there are only two entities: groups of users and one server.

References

[1] Changyu Dong, Giovanni Russello, Naranker Dulay, *Shared and Searchable Encrypted Data for Untrusted Servers*, in the 22nd IFIP WG 11.3 working conference on Data and Applications Security, 2008

[2] A.-A. Ivan and Y. Dodis, *Proxy cryptography revisited*, in the Network and Distributed System Security Symposium (NDSS) 2003

[3] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, *Public key encryption with keyword search*, in the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT) 2004.

[4] D. Boneh and M. Franklin, *Identity-based Encryption from the Weil Pairing*, in the International Cryptology Conference (CRYPTO) 2001.

[5] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, *Searchable symmetric encryption: improved definitions and efficient constructions*, in the ACM Conference on Computer and Communications Security (CCS) 2006.

[6] D. X. Song, D. Wagner, and A. Perrig, *Practical techniques for searches on encrypted data*, in S and P 2000.

[7] E. Goh, *Secure indexes*, in Cryptology ePrint Archive, Report 2003/216, 2003.