Journal of Computer Science and Cybernetics, V.35, N.3 (2019), 233–249
DOI 10.15625/1813-9663/35/3/13667

A CIPHERTEXT-POLICY ATTRIBUTE-BASED SEARCHABLE ENCRYPTION SCHEME IN NON-INTERACTIVE MODEL

VAN ANH TRINH[1], VIET CUONG TRINH[2,*]

[1] *Thanh Hoa University of Culture, Sports and Tourism*
[2] *Hong Duc University, Thanh Hoa, Viet Nam*
[*] *Trinhvietcuong@hdu.edu.vn*

Abstract. We address the problem of searching on encrypted data with expressive searching predicate and multi-writer/multi-reader, a cryptographic primitive which has many concrete application scenarios such as cloud computing, email gateway application and so on. In this paper, we propose a public-key encryption with keyword search scheme relied on the ciphertext-policy attribute-based encryption scheme. In our system, we consider the model where a user can generate trapdoors by himself/herself, we thus can remove the Trusted Trapdoor Generator which can save the resource and communication overhead. We also investigate the problem of combination of a public key encryption used to encrypt data and a public-key encryption with keyword search used to encrypt keywords, which can save the storage of the whole system.

Keywords. Attribute-based Encryption; Searchable Encryption; Searching on Encrypted Data.

1. INTRODUCTION

Searching on encrypted data is an important task, which can be applicable to many practical contexts such as cloud computing or email gateway application. In the context of cloud computing, the user's data is first encrypted and then outsourced to the cloud server. When user would like to find some specific data, he/she needs to ask help from the cloud server, user however doesn't want the cloud server to know about his/her original data. In the email gateway application, when anyone would like to securely send email to Alice, he/she encrypts the content of email under Alice's public key before sending. On the other hand, Alice would like to set priority order of receiving her emails. To this aim, Alice gives the email gateway the ability to check the priority order of incoming emails and then send to her emails in the order she wants. However, Alice also doesn't want the email gateway to know about the content of such emails.

Searchable Encryption (SE) was introduced in [2, 16] to deal with such aforementioned problems. In a nutshell, in a system which supports SE, we append encrypted keywords with corresponding encrypted data. User then relies on his/her secret key in SE scheme and chosen keywords to generate a trapdoor for the cloud server (or the email gateway) to perform the search. The trapdoor is generated in such a way that the cloud server (or the email gateway) using this trapdoor can perform successfully the search, but doesn't get any information about the original data in the resulted ciphertexts. On the other hand, since keywords are encrypted, unauthorized users (called outsiders) as well as cloud server (called

insider) ideally also don't know any information about keywords in each ciphertext. We can category SE in two types:

1. SE in the private-key setting [16], where there is only one writer (data owner who encrypts the data as well as the corresponding keywords) and one/multi reader (user who would like to search and then should be able to decrypt the resulted ciphertexts). This type of SE has obviously limited applications in practice. For example, it cannot apply to the context of sending email above since anyone should have the capacity of encrypting the content of emails sent to Alice.

2. SE in the public-key setting [2], where there are multi-writer and one/multi reader. A full searchable encryption system in practice includes two components: the first is a Public Key Encryption (PKE) scheme used to encrypt data; the second is a Public-Key Encryption with keyword Search (PEKS) used to encrypt keywords. Such full system is called a PKE-PEKS scheme. In a PKE-PEKS scheme, a full ciphertext, including both the encrypted keywords and encrypted data, should be in the form $\mathsf{PKE}_{Alice_{pk}}(\text{data})||\mathsf{PEKS}_{Alice'_{pk}}(\text{keywords})$.

   There are two cases: PKE and PEKS are independent, that means Alice's public-key/secret-key in PKE is different to the ones in PEKS; and otherwise, where Alice's public-key/secret-key could be the same in both PKE and PEKS. Obviously, such full system will become more efficient in the latter case. However, in this case we have to consider carefully the security of the full system [10] since the adversary is now more powerful than the one in the former case. When PKE and PEKS are independent, we often only care about PEKS scheme and omit the PKE scheme for simplicity. In some schemes [6, 11, 14], Alice cannot generate the trapdoor by himself/herself, he/she needs to contact with a Trusted Trapdoor Generator (TTG), that will obviously increase the communication overhead of the user, and moreover the Trusted Trapdoor Generator should be always online. We call such schemes interactive schemes.

   In summary, there are several following important properties one should take into account when estimating a system which supports searching on encrypted data:

   - Efficiency: Performance of encryption/decryption/searching algorithm, key-size/ciphertext-size, PKE and PEKS are independent or not, interactive or non-interactive, etc;
   - Expressive searching predicate: Whether or not the PEKS scheme supports conjunctive keywords or even boolean formulas of keywords for searching. Obviously, this property is more desirable than simple equality keyword search in practice;
   - Trapdoor security: Cloud server with a trapdoor in hand knows nothing about the keywords in the ciphertext and trapdoor, even when the trapdoor "matches" the ciphertext. We note that this property is very hard to achieve in the public-key setting, to the best of our knowledge there is only one scheme [6] that can *partially* achieve this property.
   - Keyword security: Ideally, unauthorized users and cloud server cannot derive any information about keywords in the ciphertext.

1.1. Related work

Over past decade, substantial progress has been made on problem of searching on encrypted data [1, 2, 3, 6, 8, 9, 11, 14, 16, 17, 18, 19], to name a few. These papers use different techniques and consider different situations for searching on encrypted data. SE in private-key setting and supports only single-writer/single-reader was first introduced in [16]. Continue this line of research, the authors in [3] investigated searchable encryption with conjunctive keyword searches and boolean queries. The authors in [17, 18] went further to investigate searchable encryption scheme in single-writer/multi-reader setting and *partially* trapdoor security. *Partially* non-interactive which can reduce the communication overhead was also investigated in [17].

SE in the public-key setting was first introduced by Boneh *et al.* [2], but their schemes only support multi-writer/single-reader and equality queries. In [1, 19], the authors extended to support multi-writer/multi-reader but their schemes still only support equality queries. Expressive searching predicate and multi-writer/multi-reader were investigated in [6, 8, 11, 12, 14], where authors manage to transform from a key policy/ciphertext policy attribute-based encryption scheme to a PEKS scheme, these schemes are thus called key policy/ciphertext policy attribute-based searchable encryption scheme. The authors in [6] went further to consider *partially* trapdoor security in the sense that, they split a keyword into two parts: The keyword name and the keyword value, where one keyword name can have many keyword values. They then showed that in their scheme, the cloud server with the trapdoor in hand can only know keyword names but nothing about keyword values in the ciphertext. This interesting property is useful in some specific practical contexts. However, the downside of this technique is that the searching time is only acceptable if the keyword names are included in the ciphertext, this leads to the fact that anyone can also know the keyword names in the ciphertext. On the other hand, the combination of PKE and PEKS was investigated in [10] where [10] also investigated the non-interactive property, however this scheme does not support expressive searching predicate.

1.2. Our contribution and organization of the paper

In this paper, we propose a PKE-PEKS scheme supporting both the expressive searching predicate and multi-writer/multi-reader, our scheme is built from the CP-ABE scheme in [13], we thus name our scheme CP-ABSE scheme for short. Our scheme has following properties:

- Our scheme is a combination of an existing PKE scheme (which is exactly the CP-ABE in [13]) and a new proposed PEKS scheme. In our scheme, user has only one pair of public key/secret key for both PKE and PEKS, and moreover user can use the CP-ABE setting to encrypt/decrypt data;

- Our scheme is non-interactive: User can generate the trapdoor by himself/herself, we thus can remove the Trusted Trapdoor Generator in our system. On the other hand, since trapdoor is generated from user's secret key, user is able to decrypt all resulted ciphertexts which can save time and communication overhead of the system;

- Efficiency: Since our CP-ABSE scheme is built from the CP-ABE scheme in [13], our scheme naturally inherits the efficiency and properties of this CP-ABE scheme such as

constant-size of user's secret key, optimized ciphertext size, multi-authority and fast decryption. Note that the CP-ABE scheme in [13] is still one of the most efficient CP-ABE schemes to date.

- We also note that our scheme does not achieve trapdoor security. We emphasize that this property is very hard to achieve in the public-key setting, to the best of our knowledge there is only one scheme [6] that can *partially* achieve this property.

In the Section 5, we give the details comparison among our scheme and several schemes which also support both the expressive searching predicate and multi-writer/multi-reader.

The paper includes 6 sections. The first section presents the definition and security model of a CP-ABSE scheme. In Section 3, we present the construction of our CP-ABSE scheme and prove that it is secure in the following section. The comparison and discussions are given in Section 5. Finally, the conclusion is in Section 6.

2. PRELIMINARIES

In this section, we first give the system workflow and the threat model of our system, then we present the definition and security model for our CP-ABSE scheme.

2.1. Ciphertext policy attribute based searchable encryption

2.1.1. System workflow and threat model

Our CP-ABSE scheme is a combination of a traditional CP-ABE scheme and a PEKS scheme supporting expressive searching predicate. In our scheme, there are four entities: data owner; user; cloud server and Private Key Generator (PKG). More precisely:

1. PKG: Play the role of PKG in traditional CP-ABE scheme, generates secret keys for users.

2. Data owner: Encrypt data as well as corresponding keywords, upload them to a public cloud.

3. User: Rely on his/her secret key to generate a trapdoor, send this trapdoor to the cloud server and get back resulted ciphertexts. Finally, decrypt resulted ciphertexts to recover data.

4. Cloud server: Receive a trapdoor from a user, perform the search based on the trapdoor and send back resulted ciphertexts to the user.

Threat model. Similar to the threat model in recent schemes [6, 8, 9, 11, 12, 14], in our system, there are two goals for which an adversary would like to achieve: getting information about encrypted data and getting information about encrypted keywords.

2.1.2. System algorithms

Formally, our CP-ABSE scheme includes seven following probabilistic algorithms.

Setup$(1^\nu, \mathcal{B})$: The inputs of this algorithm are security parameter ν and the description of attribute universe $\mathcal{B}$, the outputs are master key MSK and the public parameters param of the system.

Extract$(u, \mathcal{B}(u), \mathsf{MSK}, \mathsf{param})$: The inputs of this algorithm are attribute set $\mathcal{B}(u)$ of user u, param and MSK, the output is the user's secret key d_u.

Encrypt$(\mathcal{M}, \mathbb{A}, \mathsf{param})$: The inputs of this algorithm are param, a message $\mathcal{M}$ and an access policy $\mathbb{A}$ over the universe of attributes, the output is ciphertext ct along with a description of the access policy $\mathbb{A}$.

Decrypt$(ct, d_u, \mathsf{param})$: The inputs of this algorithm are param, the ciphertext ct and the secret key d_u of user u, the output is the message $\mathcal{M}$ if and only if $\mathcal{B}(u)$ satisfies $\mathbb{A}$. Otherwise, the output is $\perp$.

Trapdoor$(d_u, W_i, \mathsf{param})$: The inputs of this algorithm are param, secret key d_u of user u and a set of keywords user would like to search W_i, the output is the trapdoor tds.

EncryptKW$(\mathcal{KF}, \mathbb{A}', \mathsf{param})$: The inputs of this algorithm are param, an access policy $\mathbb{A}'$ over the universe of attributes and an access policy $\mathcal{KF}$ over the universe of keywords, the output is the ciphertext ct' along with a description of the access policy $\mathbb{A}'$.

Search$(\mathrm{tds}, ct', \mathsf{param})$: The inputs of this algorithm are param, a trapdoor tds and a ciphertext ct', the output is 1 if the keyword set W_i embedded in tds matches the access structure $\mathcal{KF}$ in ct' and $\mathcal{B}(u)$ satisfies $\mathbb{A}'$. Otherwise, the output is 0.

We note that the full ciphertext should be the couple (ct, ct'). In addition, in order for user to be able to decrypt the resulted ciphertext, we choose $\mathbb{A}'$ in ct' such that if $\mathcal{B}(u)$ satisfies $\mathbb{A}'$ then $\mathcal{B}(u)$ satisfies $\mathbb{A}$ in ct.

2.1.3. Security model

Selective semantic security. The selective semantic security game is similar to the one in [13], except that the adversary can ask additional corruption trapdoor query. Due to the space limitations we refer the reader to [13] for details.

Insider security. Assume that $\mathcal{A}$ is the attacker, $\mathcal{C}$ is the challenger. The insider security game is defined as follows.

Setup$(1^\nu, \mathcal{B})$. At the beginning of the game, the adversary $\mathcal{A}$ provides a target access policy $\mathbb{A}^*$ over universe of attributes, and two equal target access policy $\mathcal{KF}_0^*$, $\mathcal{KF}_1^*$ over universe of keywords for which she intends to attack, where "equal access policy" means that if $\mathcal{KF}_0^*$ and $\mathcal{KF}_1^*$ are described in the DNF form, they have the same number of clauses. $\mathcal{C}$ runs the **Setup**$(1^\nu, \mathcal{B})$ algorithm to obtain param and MSK. She then gives param to $\mathcal{A}$.

Query phase 1. $\mathcal{A}$ chooses a set of attributes $\mathcal{B}(u)$ as well as a set of keywords W_i and asks corruption trapdoor query corresponding to these sets. The challenger computes and returns corresponding tds to the adversary.

Challenge. $\mathcal{C}$ chooses $b \xleftarrow{\$} \{0,1\}$ and runs $\mathbf{EncryptKW}(\mathbb{A}^*, \mathcal{KF}_b^*, \mathsf{param})$ to generate ct'^*. Finally, $\mathcal{C}$ outputs ct'^*.

Query phase 2. The same as phase 1.

Guess. $\mathcal{A}$ finally outputs $b' \in \{0,1\}$ as its guess for b.

$\mathcal{A}$ wins the game if $b' = b$, and if $\mathcal{A}$ never asks on $\mathcal{B}(u), W_i$ such that both $\mathcal{B}(u)$ satisfies $\mathbb{A}^*$ and W_i satisfies either $\mathcal{KF}_0^*$ or $\mathcal{KF}_1^*$. The advantage of $\mathcal{A}$ to win the game is defined

$$\mathbf{Adv}_{\mathcal{A}}^{IS} = \Pr\left[b = b'\right] - \frac{1}{2}.$$

Definition 1. A ciphertext-policy attribute-based searchable encryption scheme achieves insider security if all polynomial time adversaries have at most a negligible advantage in the above game.

Outsider security. The outsider security game is similar to the insider game, the difference is that the adversary can ask corrupted secret key queries, instead of corrupted trapdoor queries.

Due to the space limitations, we refer the reader to [13] for the definitions of Access Structures, LSSS Matrices, Bilinear Maps and $(P, Q, f) - \mathsf{GDDHE}$ Assumptions and so on.

3. CIPHERTEXT POLICY ATTRIBUTE BASED SEARCHABLE ENCRYPTION

In this paper, we rely on the CP-ABE scheme in [13] to build our CP-ABSE scheme. Concretely, our CP-ABSE scheme is a combination of CP-ABE scheme in [13] and a new PEKS scheme, where the later scheme is also built from the former scheme. User in our scheme, therefore, can use the same public key and secret key for both CP-ABE scheme and PEKS scheme.

In our scheme, user relies on his/her secret key and a set of chosen keywords $W = (w_1, \ldots, w_t)$ to generate the trapdoor. More precisely, from a set of chosen keywords W, user has to indicate exactly which combinations of keywords he/she would like to search. Consider the example in [6], $W = (w_1, w_2, w_3)$ where $w_1 =$ "$Diabetes$", $w_2 =$ "$Age : 30$" and $w_3 =$ "$Weight : 150 - 200$", user has to indicate the set of combinations of keywords he/she would like to search $W_i = (w_1||w_2, w_1||w_3)$. The advantage of this point is that we can save the searching time and the communication overhead, since cloud server only needs to find and then send back ciphertexts user really wants. In other schemes [5, 6, 11, 14], user doesn't indicate exactly which combinations of keywords he/she would like to search, the cloud server thus searches on all possible combinations of keywords.

3.1. Detailed construction

Our scheme is described as follows.

Setup$(\nu, \mathcal{B})$: Denote $N = |\mathcal{B}|$ the maximal number of attributes in the system, $(p, \mathbb{G}, \mathbb{G}_T, e(\cdot,\cdot))$ a bilinear group system. The algorithm first picks a random generator $g \in \mathbb{G}$, random scalars $a, \alpha, \lambda \in \mathbb{Z}_p$, computes g^a, g^α, g^λ. The algorithm continues to generate $2N$ group elements in $\mathbb{G}$ associated with N attributes in the system $h_1, \ldots, h_N, \tilde{h}_1, \ldots, \tilde{h}_N$. Let $\mathcal{H}, \tilde{\mathcal{H}}$ be hash functions such that $\mathcal{H} : \{0,1\}^* \to \mathbb{G}$ and $\tilde{\mathcal{H}} : \mathbb{G}_T \times \{0,1\}^* \to \mathbb{Z}_p$.

Suppose that the keyword universe in the system is $W = (w_1, w_2, w_3, \ldots)$, where each $w_i \in \{0,1\}^*$. In our system the set W is unbounded, we can add any new keyword into the system at anytime. For simplicity, we omit W in the global parameters. Finally, we set the master secret key and global parameters as $\mathsf{MSK} = (g^\alpha, \lambda)$ and

$$\mathsf{param} = (g, g^a, g^\lambda, e(g,g)^\alpha, h_1, \ldots, h_N, \tilde{h}_1, \ldots, \tilde{h}_N, \mathcal{H}, \tilde{\mathcal{H}}).$$

Extract$(u, \mathcal{B}(u), \mathsf{MSK}, \mathsf{param})$: Assume $\mathcal{B}(u)$ is the attribute set of user u. The algorithm chooses $s_u \xleftarrow{\$} \mathbb{Z}_p$, then computes user u's secret key as $d_u = (d_{u_0}, d'_{u_0}, \{d_{u_i}\}_{i \in \mathcal{B}(u)}, \lambda)$, where

$$d_{u_0} = g^\alpha \cdot g^{a \cdot s_u},\ d'_{u_0} = g^{s_u},\ \{d_{u_i} = h_i^{s_u}\}_{i \in \mathcal{B}(u)}.$$

User u then keeps d_{u_0} and λ secret and publishes the rest of his/her secret key to the public domain. That means the secret key of user is of the constant-size.

Encrypt$(\mathcal{M}, \mathbb{A}, \mathsf{param})$: The inputs are a message $\mathcal{M}$, an access policy $\mathbb{A}$, as well as param. Assume that $\mathbb{A}$ is a boolean formula β and that the size of β is $|\beta|$. At first, encryptor describes β in the form of DNF access policy as $\beta = (\beta_1 \vee \cdots \vee \beta_m)$, where each β_i is a set of attributes, $i = 1, \ldots, m$.

The encryptor chooses a scalar $s \xleftarrow{\$} \mathbb{Z}_p$, then computes C, C_0 as

$$C = \mathcal{M} \cdot e(g,g)^{\alpha \cdot s},\ C_0 = g^s.$$

Next, encryptor compares between m and $|\beta|$, if $m \leq |\beta|$ he/she computes

$$C_1 = (g^a \prod_{i \in \beta_1} h_i)^s, \ldots, C_m = (g^a \prod_{i \in \beta_m} h_i)^s.$$

Else, the encryptor constructs an LSSS matrix M representing the original boolean formula β, and a map function ρ such that $(M, \rho) \in (\mathbb{Z}_p^{\ell \times n}, \mathcal{F}([\ell] \to [N]))$. She then chooses a random vector $\overrightarrow{v} = (s, y_2, \ldots, y_n) \in \mathbb{Z}_p^n$. For $i = 1, \ldots, \ell$ she computes $\lambda_i = \overrightarrow{v} \cdot M_i$, where M_i is the vector corresponding to the i'th row of M. She computes

$$C_i = g^{a.\lambda_i} h_{\rho(i)}^{-s},\ i = 1, \ldots, \ell.$$

Eventually, the output is either $ct = (C, C_0, \ldots, C_m)$ along with a description of β or $ct = (C, C_0, \ldots\ , C_\ell)$ along with a description of (M, ρ).

Decrypt$(ct, d_u, \mathsf{param})$: The decryptor u first parses the ct and checks the number of elements in ct. If it is equal to $m+1$, decryptor parses the ct as $(C_0, C_1, \dots C_m)$, then finds j such that $\beta_j \subset \mathcal{B}(u)$, and computes

$$\frac{e(C_0, d_{u_0} \prod_{i\in\beta_j} d_{u_i})}{e(d'_{u_0}, C_j)} = \frac{e(g^s, g^\alpha (g^a \prod_{i\in\beta_j} h_i)^{s_u})}{e(g^{s_u}, (g^a \prod_{i\in\beta_j} h_i)^s)} = e(g,g)^{\alpha\cdot s} = K.$$

Finally, computes $\mathcal{M} = C \cdot K^{-1}$.

Else, she defines the set $I \subset \{1, 2, \dots, \ell\}$ such that $I = \{i : \rho(i) \in \mathcal{B}(u)\}$. Let $\{\omega_i \in \mathbb{Z}_p\}_{i\in I}$ be a set of constants such that if $\{\lambda_i\}$ are valid shares of any secret s according to M then $\sum_{i\in I} \omega_i \lambda_i = s$. Note that from the relation $\sum_{i\in I} \omega_i M_i = (1, 0, \dots, 0)$ where M_i is the i-th row of the matrix M, she can determine these constants. She parses the ct as $(C_0, C_1, \dots C_\ell)$ and computes

$$e(\prod_{i\in I} C_i^{-\omega_i}, d'_{u_0}) \cdot e(C_0, d_{u_0} \prod_{i\in I} d_{u_{\rho(i)}}^{-\omega_i}) = K.$$

Then computes $\mathcal{M} = C \cdot K^{-1}$.

Trapdoor$(d_u, W_i = (\tilde{w}_{i_1}, \dots, \tilde{w}_{i_k}), \mathsf{param})$: Suppose that each $\tilde{w}_{i_j} \in \{0,1\}^*$, $j \in [k]$, is a concatenation of set of keywords, for example "$Diabetes||Age : 30$".

The user randomly chooses scalars $r_1, \dots, r_k \in \mathbb{Z}_p$, computes the trapdoor

$$\begin{aligned}\mathrm{tds} &= \left(\{\mathrm{tds}_{0,j}, \mathrm{tds}_{1,j}, \{\mathrm{tds}_{2,j,\ell}\}_{\ell\in\mathcal{B}_u}\}_{j\in[k]}, \mathrm{tds}_0,\ \{\mathrm{tds}_i\}_{i\in\mathcal{B}(u)}, \tilde{W}_i\right) \\ &= \left(\{g^\alpha g^{as_u} g^{ar_j} (g^a \mathcal{H}(\tilde{w}_{i_j}))^\lambda, g^{r_j}, \{\tilde{h}_\ell^{r_j}\}_{\ell\in\mathcal{B}_u}\}_{j\in[k]},\ g^{s_u}, \{h_i^{s_u}\}_{i\in\mathcal{B}(u)}, \tilde{W}_i\right).\end{aligned}$$

where $\tilde{W}_i$ is a short description of W_i. User then sends $(\{\mathrm{tds}_{0,j}\}_{j\in[k]}, \tilde{W}_i)$ to the cloud server, he/she publishes the rest of tds to the public domain. That means the trapdoor-size is linear in the number of combinations of keywords user would like to search.

EncryptKW$(\mathcal{KF}, \mathbb{A}', \mathsf{param},)$: Assume that access policy $\mathbb{A}' = \beta = (\beta_1 \vee \dots \vee \beta_m)$ and $\mathcal{KF} = (kf_1 \vee \dots \vee kf_{m'})$, where each β_i is a set of attributes and kf_i is a concatenation of set of keywords. Note that $\beta_i \neq \beta_j, kf_{i'} \neq kf_{j'}, \forall i, j \in [m], i', j' \in [m']$.

The encryptor picks a scalar $s \xleftarrow{\$} \mathbb{Z}_p$, then computes

$$C_0 = g^s,\ C_1 = (g^a \prod_{i\in\beta_1} h_i)^s, \dots,\ C_m = (g^a \prod_{i\in\beta_m} h_i)^s,$$

$$\tilde{C}_1 = (g^a \prod_{i\in\beta_1} \tilde{h}_i)^s, \dots,\ \tilde{C}_m = (g^a \prod_{i\in\beta_m} \tilde{h}_i)^s.$$

Next, he/she computes

$$X_i = e(g,g)^{\alpha\cdot s} \cdot e(g, g^a \mathcal{H}(kf_i))^{\lambda\cdot s},\ i = 1, \dots, m',$$

then computes

$$K_1 = \tilde{\mathcal{H}}(X_1, kf_1), \dots,\ K_{m'} = \tilde{\mathcal{H}}(X_{m'}, kf_{m'}).$$

Eventually, encryptor outputs

$$ct' = (C_0, \ldots, C_m, \tilde{C}_1, \ldots, \tilde{C}_m, K_1, \ldots, K_{m'})$$

along with a description of β.

Search$(\text{tds}, ct', \mathsf{param})$: The cloud server first finds $\ell \in [m]$ such that $\beta_\ell \subset \mathcal{B}(u)$, then computes $(X_j, Y_j),\ j = 1, \ldots, k$

$$\begin{aligned} X_j &= \frac{e(C_0, \text{tds}_{0,j} \prod_{i\in\beta_\ell} \text{tds}_i \cdot \text{tds}_{2,j,i})}{e(\text{tds}_0, C_\ell) \cdot e(\text{tds}_{1,j}, \tilde{C}_\ell)} = \frac{e(g^s, g^\alpha g^{as_u} g^{ar_j} g^{a\lambda} \mathcal{H}(\tilde{w}_{i_j})^\lambda \prod_{i\in\beta_\ell} h_i^{s_u} \tilde{h}_i^{r_j})}{e(g^{s_u}, (g^a \prod_{i\in\beta_\ell} h_i)^s) \cdot e(g^{r_j}, (g^a \prod_{i\in\beta_\ell} \tilde{h}_i)^s)} \\ &= e(g,g)^{\alpha\cdot s} \cdot e(g, g^a \mathcal{H}(\tilde{w}_{i_j}))^{\lambda\cdot s}, \\ Y_j &= \tilde{\mathcal{H}}(X_j, \tilde{w}_{i_j}). \end{aligned}$$

If there exists a pair $(i,j),\ i \in [m'],\ j \in [k]$ such that $K_i = Y_j$ then the cloud server outputs "yes". Otherwise, the cloud server outputs "no". Note that, the cloud server doesn't need to compute all pairs $(X_j, Y_j),\ j = 1, \ldots, k$, as long as he/she finds a pair $(i,j),\ i \in [m'],\ j \in [k]$ such that $K_i = Y_j$, he/she outputs "yes" and stops.

Correctness: It is easy to verify that if there exists at least one pair $\tilde{w}_{i_j} \in W_i$ and $kf_t \in \mathcal{KF}$ such that $\tilde{w}_{i_j} = kf_t$, then

$$X_t = e(g,g)^{\alpha\cdot s} \cdot e(g, g^a \mathcal{H}(kf_t))^{\lambda\cdot s} = e(g,g)^{\alpha\cdot s} \cdot e(g, g^a \mathcal{H}(\tilde{w}_{i_j}))^{\lambda\cdot s} = X_j,$$

that means

$$K_t = \tilde{\mathcal{H}}(X_t, kf_t) = \tilde{\mathcal{H}}(X_j, \tilde{w}_{i_j}) = Y_j.$$

Remark 1. As in [13] all sets β_i must be disjoint subsets to resist the simple attack, $i = 1, \ldots, m$. This leads to the fact that attributes in the system cannot be reused in the access formula. To deal with this problem, they allow each attribute to have $k_{\max}$ copies of itself as in [4, 7]. Note that the user's secret key is still of constant-size.

4. SECURITY

In this section, we show that our scheme is secure in the model defined in Subsection 2.1.3. We first refer the reader to the modified BDHE assumption defined in [13], and then we define a new modified BDHE assumption. We finally prove our scheme achieves the selective semantic security under the new modified BDHE assumption, and achieves the insider and outsider security under the modified BDHE assumption.

Definition 2. (New Modified-BDHE problem) Let $(p, \mathbb{G}, \mathbb{G}_T, e)$ be a bilinear group system, pick $a, t, s, q, \theta, r_1, \ldots, r_\theta \overset{\$}{\leftarrow} \mathbb{Z}_p$, a generator $g \in \mathbb{G}$. Given

$$\begin{aligned} \vec{Y} = \Big(g, g^s, g^a, \ldots, g^{a^q}, g^{a^{q+2}}, \ldots, g^{a^{2q}}, g^{s(at+a)}, g^{at}, \ldots, g^{a^q t}, \\ g^{a^{q+2}t}, \ldots, g^{a^{2q}t}, g^{a^{q+1}} g^{ar_1}, \ldots, g^{a^{q+1}} g^{ar_\theta}, g^{r_1}, \ldots, g^{r_\theta}\Big) \end{aligned}$$

it is hard to distinguish between $T = e(g,g)^{a^{q+1}s} \in \mathbb{G}_T$ and $T \overset{\$}{\leftarrow} \mathbb{G}_T$.

Assume that $\mathcal{A}$ is an adversary that outputs $b \in \{0,1\}$ with advantage ϵ in solving new Modified-BDHE problem in $\mathbb{G}$ if

$$\left|\Pr\left[\mathcal{A}(\vec{Y}, T = e(g,g)^{a^{q+1}s}) = 0\right] - \Pr\left[\mathcal{A}(\vec{Y}, T = R) = 0\right]\right| \geq \epsilon.$$

Definition 3. The new Modified-BDHE assumption is secure if no polytime adversary has a non-negligible advantage in solving the new Modified-BDHE problem.

Intuitively, to compute $e(g,g)^{a^{q+1}s}$ one should know one of the values $g^{a^{q+1}}$ or $g^{a^{q+1}t}$ or $e(g,g)^{sar_i},\ i \in [\theta]$, but these elements are not provided in $\vec{Y}$.

4.1. Selective semantic security

Theorem 1. *Let β^* be the challenge access policy, from β^* we construct the corresponding challenge LSSS matrix L' of size $\ell' \times n'$ and map function ρ'. We next describe $\beta^* = \beta_1^* \vee \cdots \vee \beta_m^*$ where $\beta_i^*,\ i = 1, \ldots, m$ are disjoint sets and then construct the corresponding challenge LSSS matrix L^* of size $\ell^* \times n^*$ and map function ρ^*. If those LSSS matrices satisfy $\ell', n', \ell^*, n^* \leq q$, and if $\theta \geq k^* \cdot q^*$ where k^* and q^* are maximum number of combinations of keywords in a trapdoor and maximum number of trapdoor queries corresponding to β^* adversary can make, respectively, our scheme is selectively semantic secure under the new Modified-BDHE assumption.*

Compare to the proof in [13], here the simulator needs to answer additional corruption trapdoor query. To answer this kind of query, the simulator uses the elements $g^{a^{q+1}}g^{a.r_1}, \ldots,$ $g^{a^{q+1}}g^{a.r_\theta}, g^{r_1}, \ldots\ , g^{r_\theta}$. Note that these elements only appear in new Modified-BDHE assumption, not in Modified-BDHE assumption. The rest of the proof of this theorem is similar to the one in [13].

4.2. Keyword security

4.2.1. Insider security

Theorem 2. *Assume that $\beta^* = \beta_1^* \vee \cdots \vee \beta_m^*$ is the challenge access policy and from β^* construct a corresponding challenge LSSS matrix L^* of size $\ell^* \times n^*$ and map function ρ^*. If this LSSS matrix satisfies $\ell^*,\ n^* \leq q$, our scheme achieves insider security under the Modified-BDHE assumption in the random oracle model.*

Proof

In this proof we show that the simulator $\mathcal{S}$ who attacks Modified - BDHE assumption can simulate an adversary $\mathcal{A}$ who attacks our scheme in the insider security game as defined in the Subsection 2.1.3. As a result, if $\mathcal{A}$ wins with non-negligible advantage then $\mathcal{S}$ also can win with non-negligible advantage. More precisely:

At the setup phase, $\mathcal{S}$ is given an instance of Modified-BDHE assumption, and then receives the challenge access policy $\beta^* = \beta_1^* \vee \cdots \vee \beta_m^*$ as well as $\mathcal{KF}_0^* = (kf_{0,1}^*, \ldots, kf_{0,m'}^*)$ and $\mathcal{KF}_1^* = (kf_{1,1}^*, \ldots, kf_{1,m'}^*)$ from $\mathcal{A}$. Note that $\beta_i^*,\ i = 1, \ldots, m$ are disjoint sets. From challenge access policy $\beta^* = \beta_1^* \vee \cdots \vee \beta_m^*$, simulator builds LSSS matrix $(M_{\ell^* \times n^*}^*, \rho^*)$

such that both $\ell^*,\ n^* \leq q$. To program the parameters for the system, simulator picks $\alpha' \xleftarrow{\$} \mathbb{Z}_p$ and implicitly sets $\alpha = \alpha' + a^{q+1}$, then computes $e(g,g)^{\alpha} = e(g^a, g^{a^q})e(g,g)^{\alpha'}$. The simulator finds sets of rows of matrix M^*: $I_1, \ldots, I_m$ where $\{\rho(i),\ i \in I_j\} = \beta_j^*$ (note that $I_j,\ j = 1, \ldots, m$ are disjoint sets since β_j^* are disjoint sets). Now, β^* can be rewritten as $(\wedge\rho(i))_{i\in I_1} \vee (\wedge\rho(i))_{i\in I_2} \vee \cdots \vee (\wedge\rho(i))_{i\in I_m}$.

To program $h_1, \ldots, h_N, \tilde{h}_1, \ldots, \tilde{h}_N$, the simulator implicitly defines the vector

$$\overrightarrow{y} = (t, ta, ta^2, \ldots, ta^{n^*-1})^{\perp} \in \mathbb{Z}_p^{n^*}.$$

Let $\overrightarrow{\Lambda} = (\lambda_1, \ldots,\ \lambda_{\ell^*}) = M^* \cdot \overrightarrow{y}$ be the vector shares, for $j = 1, \ldots, \ell^*$, $\lambda_j = \sum_{i\in[n^*]} M^*_{j,i} t a^{i-1}$.

He/she next finds $\{\omega_i\}_{1\leq i\leq \ell^*}$ where for all $j = 1, \ldots, m$, $\sum_{i\in I_j} \omega_i \cdot \lambda_i = t$. Note that we can find $\{\omega_i\}_{1\leq i\leq \ell^*}$ since from the property of LSSS matrix, there exists $\{\omega_i\}_{1\leq i\leq \ell^*}$ such that for all $j = 1, \ldots, m,\ \sum_{i\in I_j} \omega_i \cdot M_i^* = (1, 0, \ldots, 0)$.

For each $h_j,\ \tilde{h}_j,\ 1 \leq j \leq N$, where there exists an index $i \in [\ell^*]$ such that $j = \rho^*(i)$ (note that the function ρ^* is injective), the simulator chooses $z_j,\ \tilde{z}_j \xleftarrow{\$} \mathbb{Z}_p$ and compute: Note that the simulator knows matrix M^* and g^{ta^k} where $k \in [n^*]$ from the instance of Modified-BDHE assumption.

$$h_j = g^{z_j} \cdot g^{\omega_i \sum_{k\in[n^*]} M^*_{i,k} t a^k} = g^{z_j} \cdot g^{a\omega_i\lambda_i}; \tilde{h}_j = g^{\tilde{z}_j} \cdot g^{\omega_i \sum_{k\in[n^*]} M^*_{i,k} t a^k} = g^{\tilde{z}_j} \cdot g^{a\omega_i\lambda_i}.$$

Otherwise, the simulator chooses $z_j,\ \tilde{z}_j \xleftarrow{\$} \mathbb{Z}_p$ and computes $h_j = g^{z_j},\ \tilde{h}_j = g^{\tilde{z}_j}$. We note that $\{h_j, \tilde{h}_j\}_{j=1,\ldots,N}$ are distributed randomly due to choosing randomly $z_j,\ \tilde{z}_j$.

To program g^{λ}, the simulator implicitly sets $\lambda = -a^q$ and computes $g^{\lambda} = (g^{a^q})^{-1}$. Simulator also chooses hash functions $\mathcal{H}, \tilde{\mathcal{H}}$ and in this proof simulator models $\mathcal{H}, \tilde{\mathcal{H}}$ as random oracles. At the end of this phase, the simulator gives following param to $\mathcal{A}$

$$(g, g^a, g^{\lambda}, e(g,g)^{\alpha}, h_1, \ldots, h_N, \tilde{h}_1, \ldots, \tilde{h}_N, \mathcal{H}, \tilde{\mathcal{H}}).$$

Query phase 1. In this phase, the simulator needs to answer five types of query:

1. The hash query.
2. The corrupted trapdoor query $(\mathcal{B}_u,\ W_i = (\tilde{w}_{i_1}, \ldots, \tilde{w}_{i_k}))$ where W_i doesn't "satisfy" $\mathcal{KF}_0^*$ or $\mathcal{KF}_1^*$, that means there doesn't exist any triple (i_j, b, b') such that $\tilde{w}_{i_j} = kf^*_{b,b'}$.
3. The corrupted trapdoor query $(\mathcal{B}_u, W_i)$ where W_i "satisfies" $\mathcal{KF}_0^*$ or $\mathcal{KF}_1^*$, but $\mathcal{B}_u$ doesn't "satisfy" β^*.
4. The partially corrupted trapdoor query $(\mathcal{B}_u, W_i)$ where W_i "satisfies" $\mathcal{KF}_0^*$ or $\mathcal{KF}_1^*$ and $\mathcal{B}_u$ "satisfies" β^*. Note that user only keeps $(\{\mathrm{tds}_{0,j}\}_{j\in[k]}, \tilde{W}_i)$ secret and publishes the rest of tds to the public domain. That means $\mathcal{A}$ can know $(\{\mathrm{tds}_{1,j}, \{\mathrm{tds}_{2,j,\ell}\}_{\ell\in\mathcal{B}_u}\}_{j\in[k]}, \mathrm{tds}_0, \{\mathrm{tds}_i\}_{i\in\mathcal{B}_u})$ for any $(\mathcal{B}_u, W_i = (\tilde{w}_{i_1}, \ldots, \tilde{w}_{i_k}))$.

5. The partially corrupted secret key query $\mathcal{B}_u$ for any $\mathcal{B}_u$. The reason is that user only keeps d_{u_0} secret.

- Regarding the hash query: Simulator creates two lists $\mathcal{L}$, $\tilde{\mathcal{L}}$, at the beginning $\mathcal{L}$, $\tilde{\mathcal{L}}$ are empty. For each hash query corresponding to $\tilde{w}_i$ which doesn't satisfy $\mathcal{KF}_0^*$ or $\mathcal{KF}_1^*$, the simulator first checks whether $\tilde{w}_i$ has been queried before. If not, he/she chooses $y_i \xleftarrow{\$} \mathbb{Z}_p$ and adds triple $(\tilde{w}_i, g^{y_i}, y_i) \in (\{0,1\}^*, \mathbb{G}, \mathbb{Z}_p)$ into $\mathcal{L}$ and returns g^{y_i} to $\mathcal{A}$. Otherwise, he/she simply finds the triple $(\tilde{w}_i, g^{y_i}, y_i)$ and returns g^{y_i} to $\mathcal{A}$. In the case $\tilde{w}_i$ "satisfies" $\mathcal{KF}_0^*$ or $\mathcal{KF}_1^*$, the simulator first checks whether $\tilde{w}_i$ has been queried before. If not, he/she chooses $y_i \xleftarrow{\$} \mathbb{Z}_p$ and adds triple $(\tilde{w}_i, g^{-a} \cdot g^{y_i}, y_i)$ into $\mathcal{L}$ and returns $g^{-a} \cdot g^{y_i}$ to $\mathcal{A}$. Otherwise, he/she simply finds the triple $(\tilde{w}_i, g^{-a} \cdot g^{y_i}, y_i)$ and returns $g^{-a} \cdot g^{y_i}$ to $\mathcal{A}$. For each hash query corresponding to (K_i, kf_j) where $K_i \in \mathbb{G}_T, kf_j \in \{0,1\}^*$, simulator first checks whether (K_i, kf_j) has been queried before. If not, he/she chooses $y_{i_j} \xleftarrow{\$} \mathbb{Z}_p$ and adds triple (K_i, kf_j, y_{i_j}) into $\tilde{\mathcal{L}}$. Otherwise, he/she simply finds the triple (K_i, kf_j, y_{i_j}) and returns y_{i_j} to $\mathcal{A}$.
- Regarding the second type of query: $\mathcal{A}$ first sends $W_i = (\tilde{w}_{i_1}, \ldots, \tilde{w}_{i_k})$ and $\mathcal{B}(u)$ to simulator with the requirement that W_i doesn't satisfy $\mathcal{KF}_0^*$ or $\mathcal{KF}_1^*$. To program each $\text{tds}_{0,j}, j \in [k]$, the simulator first checks whether $\tilde{w}_{i_j}, j \in [k]$ has been queried before. If not, he/she chooses $y_{i_j} \xleftarrow{\$} \mathbb{Z}_p$ and adds triple $(\tilde{w}_{i_j}, g^{y_{i_j}}, y_{i_j})$ into $\mathcal{L}$. In both ways, simulator knows y_{i_j} from $\mathcal{L}$, and $\mathcal{H}(\tilde{w}_{i_j}) = g^{y_{i_j}}$ since W_i doesn't satisfy $\mathcal{KF}_0^*$ or $\mathcal{KF}_1^*$. Next, simulator first chooses $s_u, r_j \xleftarrow{\$} \mathbb{Z}_p$ then computes
$$\text{tds}_{0,j} = g^{\alpha'} g^{as_u} g^{ar_j} (g^{\lambda})^{y_{i_j}} = g^{\alpha} g^{as_u} g^{ar_j} (g^a \mathcal{H}(\tilde{w}_{i_j}))^{\lambda}.$$
Note that $g^{\alpha'} = g^{\alpha'} \cdot g^{a^{q+1}} \cdot g^{-a^{q+1}} = g^{\alpha} \cdot g^{a\lambda}$, since $g^{\lambda} = g^{-a^q}$. Since simulator knows $s_u, r_j, j \in [k]$, he/she can easily computes the rest of the trapdoor for any set $\mathcal{B}(u)$. Finally, simulator returns tds to $\mathcal{A}$.
- Regarding the third type of query: $\mathcal{A}$ first sends $W_i = (\tilde{w}_{i_1}, \ldots, \tilde{w}_{i_k})$ and $\mathcal{B}(u)$ to simulator with the requirement that $\mathcal{B}(u)$ doesn't satisfy β^* and W_i "satisfies" $\mathcal{KF}_0^*$ or $\mathcal{KF}_1^*$. The simulator first finds a vector $\overrightarrow{x} = (x_1, \ldots, x_{n^*}) \in \mathbb{Z}_p^{n^*}$ such that $x_1 = -1$ and for all i where $\rho^*(i) \in \mathcal{B}(u)$ the product $\langle \overrightarrow{x} \cdot M_i^* \rangle = 0$. The simulator continues to pick $\zeta \xleftarrow{\$} \mathbb{Z}_p$ and implictly define the value s_u as
$$s_u = \zeta + x_1 a^q + x_2 a^{q-1} + \cdots + x_{n^*} a^{q-n^*+1}.$$
The simulator computes
$$d_{u_0} = g^{\alpha'} g^{a\zeta} \prod_{i=2,\ldots,n^*} (g^{a^{q+1-i}})^{x_i} = g^{\alpha} \cdot g^{a \cdot s_u}; d'_{u_0} = g^{\zeta} \prod_{i=1,\ldots,n^*} (g^{a^{q+1-i}})^{x_i} = g^{s_u}.$$
For $j \in \mathcal{B}(u)$ such that there is no $i \in [\ell^*]$ satisfying $\rho^*(i) = j$. The simulator knows values z_j and computes $h_j^{s_u} = (g^{s_u})^{z_j}$. For $j \in \mathcal{B}(u)$ such that there is an indice $i \in [\ell^*]$ satisfying $\rho^*(i) = j$. The simulator computes
$$h_j^{s_u} = (g^{s_u})^{z_j} \cdot g^{(\zeta + x_1 a^q + \cdots + x_{n^*} a^{q-n^*+1}) \omega_i \sum_{k \in [n^*]} M_{i,k}^* t a^k}.$$

Note that the product $\langle \overrightarrow{x} \cdot M_i^* \rangle = 0$ thus the simulator doesn't need to know the unknown term of form $g^{a^{q+1}t}$ to compute $h_j^{s_u}$, all other terms he/she knows from the assumption. Simulator simply sets g^{s_u} and $\{h_j^{s_u}\}_{j \in \mathcal{B}_u}$ as tds_0 and $\{\mathrm{tds}_i\}_{i \in \mathcal{B}_u}$, respectively. To program $\{\mathrm{tds}_{0,j}, \mathrm{tds}_{1,j}, \{\mathrm{tds}_{2,j,\ell}\}_{\ell \in \mathcal{B}_u}\}_{j \in [k]}$, the simulator considers two cases:

1. $\tilde{w}_{i_j}$ "satisfies" $\mathcal{KF}_0^*$ or $\mathcal{KF}_1^*$, that means there exists at least a triple (i_j, b, b') such that $\tilde{w}_{i_j} = kf_{b,b'}^*$. Simulator checks whether $\tilde{w}_{i_j}$ has been queried before. If not, he/she chooses $y_{i_j} \xleftarrow{\$} \mathbb{Z}_p$ and adds triple $(\tilde{w}_{i_j}, g^{-a} g^{y_{i_j}}, y_{i_j})$ into $\mathcal{L}$. In both ways, simulator knows y_{i_j} from $\mathcal{L}$, and $\mathcal{H}(\tilde{w}_{i_j}) = g^{-a} g^{y_{i_j}}$. Next, simulator chooses $r_j \xleftarrow{\$} \mathbb{Z}_p$ then computes

$$\mathrm{tds}_{0,j} = g^{\alpha} g^{a s_u} g^{a r_j} (g^{-a^q})^{y_{ij}} = g^{\alpha} g^{a s_u} g^{a r_j} (g^a \mathcal{H}(\tilde{w}_{i_j}))^{\lambda}.$$

   Note that $\lambda = -a^q$. With known r_j, simulator can easily compute $\mathrm{tds}_{1,j}$, $\{\mathrm{tds}_{2,j,\ell}\}_{\ell \in \mathcal{B}_u}$.
2. $\tilde{w}_{i_j}$ doesn't "satisfy" $\mathcal{KF}_0^*$ or $\mathcal{KF}_1^*$. Simulator checks whether $\tilde{w}_{i_j}$ has been queried before. If not, he/she chooses $y_{i_j} \xleftarrow{\$} \mathbb{Z}_p$ and adds triple $(\tilde{w}_{i_j}, g^{y_{i_j}}, y_{i_j})$ into $\mathcal{L}$. In both ways, simulator knows y_{i_j} from $\mathcal{L}$, and $\mathcal{H}(\tilde{w}_{i_j}) = g^{y_{i_j}}$. Next, simulator picks $\zeta_j \xleftarrow{\$} \mathbb{Z}_p$ and implicitly defines the value r_j as

$$r_j = \zeta_j + x_1 a^q + x_2 a^{q-1} + \cdots + x_{n^*} a^{q-n^*+1},$$

   then similarly computes $g^{\alpha} g^{a r_j}, g^{r_j}, \{\tilde{h_\ell}^{r_j}\}_{\ell \in \mathcal{B}_u}$ as above (note that r_j now plays the role as s_u). He/she then sets $g^{-r_j}, \{\tilde{h_\ell}^{-r_j}\}_{\ell \in \mathcal{B}_u}$ as $\mathrm{tds}_{1,j}, \{\mathrm{tds}_{2,j,\ell}\}_{\ell \in \mathcal{B}_u}$ respectively, and computes

$$\mathrm{tds}_{0,j} = g^{\alpha} g^{a s_u} g^{-\alpha} g^{-a r_j} g^{\alpha'} (g^{-a^q})^{y_{i_j}} = g^{\alpha} g^{a s_u} g^{-a r_j} (g^a \mathcal{H}(\tilde{w}_{i_j}))^{\lambda}.$$

   Note that $g^{\alpha} = g^{\alpha'} g^{a^{q+1}}$ and $(g^a \mathcal{H}(\tilde{w}_{i_j}))^{\lambda} = g^{-a^{q+1}} (g^{-a^q})^{y_{i_j}}$.

Finally, simulator returns tds to $\mathcal{A}$.

- Regarding the fourth and fifth types of query: It is straightforward since the unknown value g^{α} only appears in the $\mathrm{tds}_{0,j}$ and d_{u_0}, therefore simulator can simply choose $s_u, \{r_j\}_{j \in [k]} \xleftarrow{\$} \mathbb{Z}_p$ and then computes tds, or choose $s_u, \xleftarrow{\$} \mathbb{Z}_p$ and then computes d_u.

Challenge: The simulator picks a random bit b, computes $C_0^* = g^s$ and $(C_1^*, \ldots, C_m^*) = I$, $(\tilde{C}_1^*, \ldots, \tilde{C}_m^*) = J$, where

$$I = \left(g^{s(a+at)} g^{\sum_{i \in I_1} sz_{\rho^*(i)}}, \ldots, g^{s(a+at)} g^{\sum_{i \in I_m} sz_{\rho^*(i)}} \right)$$
$$= \left(g^s, (g^a \prod_{i \in \beta_1^*} h_i)^s, \ldots, (g^a \prod_{i \in \beta_m^*} h_i)^s \right),$$

$$J = \left(g^{s(a+at)} g^{\sum_{i\in I_1} s\tilde{z}_{\rho^*(i)}}, \ldots, g^{s(a+at)} g^{\sum_{i\in I_m} s\tilde{z}_{\rho^*(i)}}\right)$$
$$= \left((g^a \prod_{i\in\beta_1^*} \tilde{h}_i)^s, \ldots, (g^a \prod_{i\in\beta_m^*} \tilde{h}_i)^s\right).$$

To computes $\{K_i^*\}_{i\in[m']}$, simulator first checks whether $kf_{b,i}^*$ has been queried before. If not, he/she chooses $y_i \xleftarrow{\$} \mathbb{Z}_p$ and adds triple $(kf_{b,i}^*, g^{-a}g^{y_i}, y_i)$ into $\mathcal{L}$. Otherwise, he/she finds $(kf_{b,i}^*, g^{-a}g^{y_i}, y_i)$ from $\mathcal{L}$. In both ways, simulator knows y_i from $\mathcal{L}$, and $\mathcal{H}(kf_{b,i}^*) = g^{-a}g^{y_i}$. He/she computes

$$X_i^* = T \cdot e(g^s, g^{\alpha'}) \cdot e(g^s, (g^\lambda)^{y_i}) = T \cdot e(g^s, g^{\alpha'}) \cdot e(g^\lambda, g^a \mathcal{H}(kf_{b,i}^*))^s$$

then computes $K_i^* = \tilde{\mathcal{H}}(X_i^*, kf_{b,i}^*)$. Finally, he/she outputs

$$ct'^* = (C_0^*, \{C_i^*\}_{i\in[m]}, \{\tilde{C}_i^*\}_{i\in[m]}, \{K_i^*\}_{i\in[m']}).$$

Note that if $T = e(g,g)^{a^{q+1}s}$ then ct'^* is in valid form.

Query phase 2. Similar to phase 1.

Guess: $\mathcal{A}$ gives his guess b' to $\mathcal{S}$, $\mathcal{S}$ will output its guess 0 corresponding to $T = e(g,g)^{a^{q+1}s}$ if $b' = b$; otherwise, $\mathcal{S}$ outputs its guess 1 corresponding to T is a random element. When $T = e(g,g)^{a^{q+1}s}$, $\mathcal{S}$ gives a perfect simulation so

$$\Pr\left[\mathcal{S}(\vec{Y}, T = e(g,g)^{a^{q+1}s}) = 0\right] = \frac{1}{2} + \mathbf{Adv}_{\mathcal{A}}^{IS}.$$

When T is a random element $\{K_i^*\}_{i\in[m']}$ are completely hidden from the $\mathcal{A}$, so $Pr[\mathcal{S}(\vec{Y}, T = R) = 0] = \frac{1}{2}$. As a result, $\mathcal{S}$ is able to play the Modified - BDHE game with non-negligible advantage (equal to $\mathbf{Adv}_{\mathcal{A}}^{IS}$) or $\mathcal{S}$ is able to break the security of Modified-BDHE assumption.

■

Outsider Security. Outsider security is similar to the case of Insider Security, due to the space limitations we omit it here.

5. COMPARISON

Regarding PEKS scheme supporting both expressive searching predicate and multi-writer /multi-reader, to our knowledge [6, 8, 9, 11, 14] are the best schemes to date. In these schemes, the authors in [9] proposed a PEKS schemes scheme with constant size of ciphertext, however their scheme only supports limited AND-gates access policy. The authors in [6, 8, 11, 14] managed to transform from existing KP-ABE or CP-ABE schemes to PEKS schemes, these schemes enjoy some interesting properties such as fast keyword search or outsourcing decryption or *partially* trapdoor security. Other properties such as fined-grain access policy, public key-size, ciphertext-size are similar to ones in our scheme, but our scheme has constant

size of secret key while they haven't, moreover our model is different to their model. We give in Fig 1 the comparison between our model and the model in [6, 8, 11, 14]. We can easily see from Fig 1 that there is no TTG in our model, user relies solely on his/her secret key to generate a trapdoor. In contrast, in their model, TTG takes charge of generating trapdoors and therefore should be always online. Compare to their model, our model has two following advantages:

- There is no TTG in our model, we therefore can save the system resource and the communication overhead between user and TTG.

- In our model, user uses his/her secret key to generate trapdoors, cloud server relies on such trapdoors to search corresponding ciphertexts, user therefore is able to decrypt all the resulted ciphertexts. In contrast, in their model, TTG takes charge of generating trapdoors and user's secret key is not involved in this process, this leads to the fact that the resulted ciphertexts may contain ciphertexts for which the user cannot decrypt. We argue that our model is more useful in practice since it is waste time and communication overhead if user receives the ciphertexts for which he/she cannot decrypt.

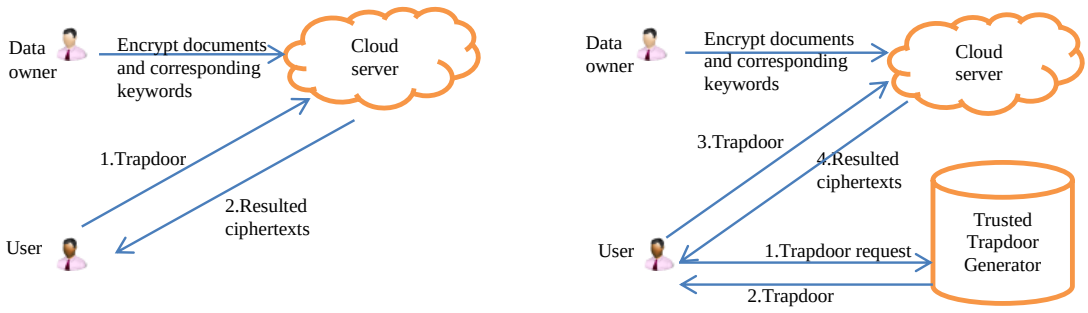

Figure 1. Our model on the left and their model on the right

We also note that the scheme in [15] can deal with well the problem of trapdoor security, and moreover this scheme is very efficient in term of both communication and computation. However, this scheme is in the different type to our scheme since our scheme supports expressive searching while the scheme in [15] supports equality queries. Consider the example in [6], $W = (w_1, w_2, w_3)$ where $w_1 =$ "$Diabetes''$, $w_2 =$ "$Age : 30''$ and $w_3 =$ "$Weight : 150 - 200''$, user in our scheme can search for ciphertexts which has keyword either "$Diabetes''$ or "$Weight : 150 - 200''$. While the user in the scheme in [15] must specify the keyword search is "$Diabetes$" or "$Weight : 150 - 200''$ and then receives only the ciphertext corresponding to the keyword search. For example, if the keyword search is "$Diabetes''$, user only receives the ciphertext corresponding to "$Diabetes''$. This is similar to the difference between traditional public key encryption and attribute-based encryption.

6. CONCLUSION

In this paper, we propose a CP-ABSE scheme which supports both expressive searching predicate and multi-writer/multi-readers. To our knowledge, our scheme has some interesting properties such as constant-size of secret key and in the non-interactive model. Our scheme will become very efficient when the number of combinations of keywords to which a user would like to search is small. Our scheme is therefore very suitable for a large class of applications in practice for which the aforementioned case falls into.

ACKNOWLEDGMENTS

This research is funded by Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number 102.01-2018.301.

REFERENCES

[1] M. R. Asghar, G. Russello, B. Crispo, and M. Ion, "Supporting complex queries and access policies for multi-user encrypted databases," in *Proceeding CCSW '13 Proceedings of the 2013 ACM workshop on Cloud computing security workshop*, Berlin, Germany, November 4, 2013, pp 77–88. Doi 10.1145/2517488.2517492

[2] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," *Advances in Cryptology - EUROCRYPT 2004 International Conference on the Theory and Applications of Cryptographic Techniques*, Interlaken, Switzerland, May 2-6, 2004, pp 506–522.

[3] D. Cash, S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for Boolean queries", in *Advances in Cryptology CRYPTO 2013 33rd Annual Cryptology Conference, Proceedings, Part I*, Santa Barbara, CA, USA, August 18-22, 2013.

[4] S. Canard and V. C. Trinh, "Constant-size ciphertext attribute-based encryption from multi-channel broadcast encryption," *Information Systems Security. 12th International Conference, ICISS 2016, Proceedings,* Jaipur, India, December 16-20, 2016.

[5] H. Cui, R. Deng, J. Liu, and Y. Li, "Attribute-based encryption with expressive and authorized keyword search," *Information Security and Privacy. 22nd Australasian Conference, ACISP 2017, Proceedings,* Auckland, New Zealand, July 35, 2017, Part I, pp. 106–126.

[6] H. Cui, Z. Wan, R. Deng, G. Wang, and Y. Li, "Efficient and expressive keyword search over encrypted data in the cloud," *IEEE Transactions on Dependable and Secure Computing,* vol. 15, no. 3, pp. 409–422, 2018. Doi 10.1109/TDSC.2016.2599883

[7] S. Hohenberger and B. Waters, "Attribute-based encryption with fast decryption," *Public-Key Cryptography PKC 2013. 16th International Conference on Practice and Theory in Public-Key Cryptography, Proceedings,* Nara, Japan, February 26 March 1, 2013, pp. 162–179.

[8] Jianting Ning, Zhenfu Cao, Xiaolei Dong, Kaitai Liang, Hui Ma, Lifei Wei, "Auditable σ-Time Outsourced Attribute-Based Encryption for Access Control in Cloud Computing," *IEEE Transactions on Information Forensics and Security,* vol. 13, no. 1, pp. 94–105, 2018. Doi 10.1109/TIFS.2017.2738601

[9] Jinguang Han, Ye Yang, Joseph K. Liu, Jiguo Li,Kaitai Liang,Jian Shen, "Expressive attribute-based keyword search with constant-size ciphertext," *In Soft Computing Journal*, vol. 22, no. 15, pp 5163-5177, August 2018.

[10] A. Kiayias, O. Oksuz, A. Russell, Q. Tang, and B. Wang, "Efficient encrypted keyword search for multi-user data sharing," *Computer Security ESORICS 2016. 21st European Symposium on Research in Computer Security, Proceedings*, Heraklion, Greece, September 26-30, 2016, Part I, pp 173–195.

[11] J. Lai, X. Zhou, R. H. Deng, Y. Li, and K. Chen, "Expressive search on encrypted data," *ASIA CCS '13 Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security,* Hangzhou, China, May 8-10, 2013, pp 243–251.

[12] Z. Lv, C. Hong, M. Zhang, and D. Feng, "Expressive and secure searchable encryption in the public key setting," *Information Security 17th International Conference, ISC 2014, Proceedings*, Hong Kong, China, October 12-14, 2014, pp 364–376.

[13] Q. M. Malluhi, A. Shikfa, and V. C. Trinh, "A ciphertext-policy attribute-based encryption scheme with optimized ciphertext size and fast decryption," *Proceeding ASIA CCS '17 Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security,* Abu Dhabi, United Arab Emirates, April 02 - 06, 2017, pp. 230–240.

[14] R. Meng, Y. Zhou, J. Ning, K. Liang, J. Han, and W. Susilo, "An efficient key-policy attribute-based searchable encryption in prime-order groups," *Provable Security 11th International Conference, ProvSec 2017, Proceedings,* Xi'an, China, October 23-25, 2017, pp. 39–56.

[15] Qiong Huang and Hongbo Li, "An efficient public-key searchable encryption scheme secure against inside keyword guessing attacks," *Information Sciences Journal*, vol. 403–404, pp. 1–14, September 2017. Doi.org/10.1016/j.ins.2017.03.038

[16] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," *Proceeding 2000 IEEE Symposium on Security and Privacy*, Berkeley, CA, USA, May 14–17, 2000, pp. 44–55. Doi 10.1109/SECPRI.2000.848445

[17] S. Sun, J. K. Liu, A. Sakzad, R. Steinfeld, and T. H. Yuen, "An efficient non-interactive multi-client searchable encryption with support for boolean queries," *Computer Security ESORICS 2016. 21st European Symposium on Research in Computer Security, Proceedings, Part I*, Heraklion, Greece, September 26–30, 2016, pp. 154–172.

[18] Y. Wang, J. Wang, S. Sun, J. Liu, W. Susilo, and X. Chen, "Towards multi-user searchable encryption supporting boolean query and fast decryption," *Provable Security 11th International Conference, ProvSec 2017, Proceedings,* Xi'an, China, October 23–25, 2017, pp. 24–38.

[19] Y. Yang, H. Lu, and J. Weng, "Multi-user private keyword search for cloud computing," *2011 IEEE Third International Conference on Cloud Computing Technology and Science,* Athens, Greece, 29 Nov.- Dec. 01, 2011, pp. 264–271. Doi 10.1109/CloudCom.2011.43

Received on March 05, 2019
Revised on April 18, 2019