Privacy-Preserving Key-Updatable Public Key Encryption with Keyword Search Supporting Ciphertext Sharing Function

Fen Wang[1], Yang Lu[1*], Zhongqi Wang[2] and Jinmei Tian[1]
[1] School of Computer and Electronic Information, Nanjing Normal University
Nanjing, Jiangsu 210046 - China
[2]Graduate School of Science and Technology, University of Tsukuba
Tsukuba, Ibaraki 305-8577 - Japan
[e-mail: shisan_x@163.com, luyangnsd@163.com, zqwang1028@outlook.com, tianjinnmei@163.com]
[*]Corresponding author: Yang Lu

Received April 6, 2021; revised June 8, 2021; accepted January 4, 2022; published January 31, 2022

Abstract

Public key encryption with keyword search (PEKS) allows a user to make search on ciphertexts without disclosing the information of encrypted messages and keywords. In practice, cryptographic operations often occur on insecure devices or mobile devices. But, these devices face the risk of being lost or stolen. Therefore, the secret keys stored on these devices are likely to be exposed. To handle the key exposure problem in PEKS, the notion of key-updatable PEKS (KU-PEKS) was proposed recently. In KU-PEKS, the users' keys can be updated as the system runs. Nevertheless, the existing KU-PEKS framework has some weaknesses. Firstly, it can't update the keyword ciphertexts on the storage server without leaking keyword information. Secondly, it needs to send the search tokens to the storage server by secure channels. Thirdly, it does not consider the search token security. In this work, a new PEKS framework named key-updatable and ciphertext-sharable PEKS (KU-CS-PEKS) is devised. This novel framework effectively overcomes the weaknesses in KU-PEKS and has the ciphertext sharing function which is not supported by KU-PEKS. The security notions for KU-CS-PEKS are formally defined and then a concrete KU-CS-PEKS scheme is proposed. The security proofs demonstrate that the KU-CS-PEKS scheme guarantees both the keyword ciphertext privacy and the search token privacy. The experimental results and comparisons bear out that the proposed scheme is practicable.

Keywords: Public key encryption with keyword search, key exposure, key update, ciphertext sharing.

This work was supported by the National Natural Science Foundation of China [61772009, 61972095, 62072104], the Natural Science Foundation of Jiangsu Province [BK20181304].

http://doi.org/10.3837/tiis.2022.01.015 **ISSN : 1976-7277**

1. Introduction

In the wake of the rapid universal application of cloud storage, a growing number of enterprises and users are choosing to preserve the data in the cloud. The cloud provides users with more efficient and flexible data services while reducing the cost of data storage. Nevertheless, the cloud storage servers are untrusted for the data owners. Therefore, the data owners may choose to encrypt the data and upload the ciphertexts to the cloud storage server. But as a result, users face the question of how to retrieve the ciphertexts. The user can choose to download all ciphertexts to the local storage, and retrieve the data plaintexts after decryption. This method is obviously inefficient. It not only requires high communication and computation overhead, but also takes up a lot of local storage space. Another method is that the user sends the storage server the decryption key, then the storage server decrypts the ciphertexts and searches on the plaintexts, which obviously loses the meaning of encryption.

In order to handle the ciphertext retrieval issue, searchable encryption was proposed [1]. Searchable encryption allows users to retrieve the ciphertexts by keywords without revealing any information about the plaintexts. It can be implemented over symmetric encryption or public key encryption. In 2000, Song *et al*. [1] first brought a symmetric searchable encryption scheme. Subsequently, many improved symmetric searchable encryption schemes [2-6] were devised. Although these symmetric searchable encryption schemes have high execution efficiency, they encounter the difficulty in key distribution. The mechanism of public key encryption with keyword search (PEKS), invented by Boneh *et al*. [7], allows a ciphertext receiver to authorize a storage server to verify whether the ciphertexts sent to them contain specific keyword(s). In PEKS, a user sends the encrypted data to a storage server. If a recipient intends to get some data ciphertexts that involve a specific keyword w, it can produce a search token T_w of the keyword w by its own secret key. Then the recipient sends the storage server the search token T_w. Once receiving T_w, the storage server can use the search token to match the ciphertexts without decryption. In the end, the storage server sends the retrieved ciphertexts to the recipient. After the concept of PEKS was put forward, various PEKS schemes and variants have been proposed [8-19]. In PEKS, the search tokens need to be conveyed over secure channels to the storage server. If the search tokens are captured by the attacker, then the indistinguishability of the keyword ciphertexts will be broken. Nevertheless, it is costly to establish a secure channel, especially over an open network. In [20], Beak *et al*. introduced the framework of secure channel free PEKS (SCF-PEKS) which does not require transmitting the search tokens through secure channels. In SCF-PEKS, the user needs to use both the public keys of the recipient and the designated storage server to encrypt the keyword to create the keyword ciphertext. Thus, only the designated server can execute the search operation by its secret key. Due to the characteristic of no secure channel, several SCF-PEKS schemes have also been presented [21-26]. Apart from this, some works [27-33] introduced PEKS into other public key cryptosystems.

The security of a cryptosystem mainly depends on the confidentiality of secret keys. Once a secret key is leaked, the security of entire cryptosystem may not be guaranteed. However, cryptographic computations are often performed on some relatively insecure devices which cannot guarantee the secrecy of secret keys. Therefore, key disclosure seems to be inevitable. To overcome the key disclosure issue in PEKS setting, Anada *et al*. [34] put froward the concept of key-updatable PEKS (KU-PEKS). They also gave a generic KU-PEKS construction that combines a PEKS scheme with a public key encryption (PKE) scheme. In KU-PEKS, each user can update his/her public and secret keys as the system runs. Once re-keying occurs, the cloud server refreshes the ciphertexts stored on it by the decrypt-then-encrypt operations.

Up to now, KU-PEKS is the only framework supporting the key update function in PEKS setting.

1.1 The Motivation and Contributions

Although KU-PEKS realizes the key update function in PEKS, it suffers from three security weaknesses.

1) KU-PEKS does not provide privacy-preserving ciphertext update function on the storage server. As we know, it is required that a keyword search encryption scheme should protect the keyword privacy. However, the KU-PEKS scheme proposed in [34] completely exposes the keyword information to the storage server when implementing ciphertext update. To update the ciphertexts, the storage server in KU-PEKS should first decrypt the old ciphertexts to get the encrypted keywords, and then re-encrypt the keywords using the new public key to produce the new keyword ciphertexts. Therefore, KU-PEKS fails in protecting the privacy of keywords.

2) KU-PEKS should transmit the search tokens secretly. In order to successfully retrieve the desired ciphertext, a user in KU-PEKS should send a search token to the storage server. However, the user should use a secure channel to transmit the search token. If not, the search token may be intercepted by an adversary and then is used to break the keyword ciphertext indistinguishability. Therefore, KU-PEKS is not suitable for the scenarios where establishing a secure channel is difficult or impossible.

3) KU-PEKS did not consider the search token indistinguishability. As Rhee *et al*. pointed out in [24], a PEKS scheme should meet the indistinguishability of keyword ciphertext/search token. If a PEKS scheme does not have the search token indistinguishability, it can't endure the keyword guessing attack by the outside attacker (OUT-KGA). In [34], Anada *et al*. constructed the KU-PEKS scheme by integrating a PEKS scheme into a PKE scheme. Unfortunately, the vast majority of PEKS schemes were not proven to be search token indistinguishable. Therefore, the KU-PEKS scheme proposed in [34] is fragile to the OUT-KGA.

In this work, a privacy-preserving key-updatable public key searchable encryption scheme is devised. The presented scheme surmounts the weaknesses in the existing KU-PEKS framework and supports the ciphertext sharing function that is not considered in Anada *et al*.'s KU-PEKS scheme. Specifically, the contributions are as below:

1) A new PEKS framework called key-updatable and ciphertext-sharable PEKS (KU-CS-PEKS) is proposed. Compared with KU-PEKS, the proposed framework enjoys the following three merits. Firstly, KU-CS-PEKS provides privacy-preserving ciphertext update function on the storage server. To update the ciphertexts without decryption, proxy re-encryption [36–37] is incorporated into the framework. Since the storage server is absolutely ignorant of the keywords encrypted in the keyword ciphertexts during ciphertext updating, KU-CS-PEKS effectively protects the privacy of the keywords. Secondly, KU-CS-PEKS removes the requirement of secure channel in search token transmission. KU-CS-PEKS employs a designated storage server to perform the match-testing operation by its secret key. Since the outside attacker cannot run the testing operation, it cannot break the indistinguishability of keyword ciphertext even if it intercepts the corresponding search token. Therefore, a public channel can be used to transmit the search tokens. Thirdly, KU-CS-PEKS offers the ciphertext sharing function. In KU-PEKS, only the data owner can access the ciphertexts. However, in practice, the data owner often needs to share his/her ciphertexts to other users. In KU-CS-PEKS, by authorizing the storage server to act as a re-encryption proxy, the data owner can share his/her ciphertexts with others. In this way, the authorized users also can access and retrieve the re-encrypted ciphertexts on the storage server. After giving the framework of KU-

CS-PEKS, the security model of KU-CS-PEKS is formalized, which captures both the indistinguishability of keyword ciphertext and search token.

2) A concrete KU-CS-PEKS scheme is developed. The security proofs show that: i) under the hardness assumption of the bilinear Diffie-Hellman inversion (BDHI) problem [38], it satisfies the original ciphertext indistinguishability and the re-encryption ciphertext indistinguishability; ii) under the hardness assumption of the hash Diffie-Hellman (HDH) problem [39], it satisfies the data owner's search token indistinguishability and the authorized user's search token indistinguishability. In addition, the KU-CS-PEKS scheme is compared with the existing KU-PEKS scheme to show its merits and applicability.

2. Preliminaries

In this section, some preliminaries that used in the paper are reviewed.

Assuming that G and G_T denote two multiplicative cyclic groups both with a prime order p, g denotes a random generator of G, and e: $G \times G \rightarrow G_T$ is a bilinear map satisfying the following characteristics:

(1) Bilinearity: $\forall m, n \in G$ and $x, y \in Z_p^*$, $e(m^x, n^y) = e(m, n)^{xy}$.

(2) Non-degeneracy: $\exists m, n \in G$, $e(m, n) \neq 1$.

(3) Computability: $\forall m, n \in G$, $e(m, n)$ can be calculated by an efficient algorithm.

Definition 1. The BDHI problem is: Inputting two elements (g, g^{α}), to calculate $e(g, g)^{1/\alpha}$, where $\alpha \in Z_q$.

Definition 2. The HDH problem is: Inputting four elements $(g, g^{\alpha}, g^{\beta}, H(g^{\gamma})) \in G \times G \times G \times \{0, 1\}^{hlen}$ and a hash function H: $G \rightarrow \{0, 1\}^{hlen}$, to output "1" if $\alpha\beta = \gamma$ or "0" otherwise, where $\alpha, \beta, \gamma \in Z_p^*$ and $hlen \in Z^+$.

3. Framework and security model of KU-CS-PEKS

3.1 Framework definition

A KU-CS-PEKS scheme includes four entities: a global parameter generator (GPG), a designated storage server, a data owner and an authorized user. The GPG is responsible for producing the global parameters for the whole system. The data owner produces the keyword ciphertext as well as the data ciphertext, then transmits the keyword ciphertext and the data ciphertext to the designated storage server. The data owner also generates both the update key and the re-encryption key which are transmit to the designated storage server. When receiving an update key or a re-encryption key, the designated storage server can update or re-encrypt the ciphertexts. When the designated storage server gets a search token from the data owner or the authorized user, it seeks out and returns all matching data ciphertexts.

Definition 3. A KU-CS-PEKS scheme is specified by ten algorithms that are shown below:

(1) $GlobalSetup(\lambda)$: Given as input a security parameter λ, the GPG executes the algorithm and outputs a set of global parameters gp.

(2) $KeyGen_{server}(gp)$: Given gp, the designated storage server executes this algorithm to produce a pair of public/secret keys (PK_S, SK_S).

(3) $KeyGen_{user}(gp)$: Given gp, the user (either a data owner or an authorized user) executes the algorithm to produce a pair of public/secret keys (PK_U, SK_U). In the following paper, the public/secret key pairs of a data owner in two different time periods i and j are denoted by

($PK_{DO,i}$, $SK_{DO,i}$) and ($PK_{DO,j}$, $SK_{DO,j}$) respectively, the public/secret key pair of an authorized user is (PK_{AU}, SK_{AU}) .

(4) *KWCiphertextGen*(gp, w, PK_S, $PK_{DO,i}$): Given gp, a keyword w, the designated storage server's public key PK_S and the data owner's public key $PK_{DO,i}$, the data owner executes this algorithm to produce a keyword ciphertext $CT^{w}_{DO,i}$ which is valid only in the time period i.

(5) *UpdateKeyGen*(gp, $SK_{DO,i}$, $SK_{DO,j}$): Given gp, two secret keys $SK_{DO,i}$ and $SK_{DO,j}$ in two different time periods i and j, the data owner executes this algorithm to produce an update key $uk_{i\to j}$. The update key $uk_{i\to j}$ is then sent to the designated storage server.

(6) *CiphertextUpdate*(gp, $uk_{i\to j}$, $CT^{w}_{DO,i}$): Given gp, an update key $uk_{i\to j}$ and a keyword ciphertext $CT^{w}_{DO,i}$ in the time period i, the designated storage server executes this algorithm to produce an update ciphertext $CT^{w}_{DO,j}$ for the time period j.

(7) *ReKeyGen*(gp, $SK_{DO,j}$, SK_{AU}): Given gp, the data owner's secret key $SK_{DO,j}$ and the authorized user's secret key SK_{AU}, the data owner and the authorized user executes this algorithm to produce a re-encryption key $rk_{DO\to AU}$ in an interactive manner.

(8) *CiphertextShare*(gp, $CT^{w}_{DO,j}$, $rk_{DO\to AU}$): Given gp, a keyword ciphertext $CT^{w}_{DO,j}$ and a re-encryption key $rk_{DO\to AU}$, the designated storage server executes this algorithm to produce a shared keyword ciphertext CT^{w}_{AU} . Note that the keyword ciphertext CT^{w}_{AU} is encrypted under the authorized user's public key PK_{AU} and the designated storage server's public key PK_S.

(9) *SearchTokenGen*(gp, w' , PK_S, SK_U): Given gp, a keyword w' , the designated storage server's public key PK_S and a secret key SK_U, the data owner or the authorized user executes the algorithm to produce a search token $T_{w'}$, where SK_U is either $SK_{DO,j}$ or SK_{AU}.

(10) *Test*(gp, CT_w, $T_{w'}$, SK_S): Given gp, a keyword ciphertext CT_w, a search token $T_{w'}$ and the designated storage server's secret key SK_S, the designated storage server executes this algorithm to output 1 if CT_w matches $T_{w'}$ (*i.e.*, $w = w'$) or 0 else, where CT_w is either $CT^{w}_{DO,j}$ or CT^{w}_{AU} .

If the following formulas are satisfied for any keyword w, then a KU-CS-PEKS scheme is correct.

(1) If $gp \leftarrow GlobalSetup(\lambda)$, $(PK_S, SK_S) \leftarrow KeyGen_{server}(gp)$, $(PK_{DO,i}, SK_{DO,i}) \leftarrow KeyGen_{user}(gp)$, $(PK_{DO,j}, SK_{DO,j}) \leftarrow KeyGen_{user}(gp)$, $CT^{w}_{DO,i} \leftarrow KWCiphertextGen(gp, w, PK_S, PK_{DO,i})$, $uk_{i\to j} \leftarrow UpdateKeyGen(gp, SK_{DO,i}, SK_{DO,j})$, $CT^{w}_{DO,j} \leftarrow CiphertextUpdate(gp, uk_{i\to j}, CT^{w}_{DO,i})$, $rk_{DO\to AU} \leftarrow ReKeyGen(gp, SK_{DO,j}, SK_{AU})$, $CT^{w}_{AU} \leftarrow CiphertextShare(gp, CT^{w}_{DO,j}, rk_{DO\to AU})$, $T_w \leftarrow SearchTokenGen(gp, w, PK_S, SK_{DO,j})$, then $1 \leftarrow Test(gp, CT^{w}_{DO,j}, T_w, SK_S)$;

(2) If $gp \leftarrow GlobalSetup(\lambda)$, $(PK_S, SK_S) \leftarrow KeyGen_{server}(gp)$, $(PK_{DO,i}, SK_{DO,i}) \leftarrow KeyGen_{user}(gp)$, $(PK_{DO,j}, SK_{DO,j}) \leftarrow KeyGen_{user}(gp)$, $(PK_{AU}, SK_{AU}) \leftarrow KeyGen_{user}(gp)$, $CT^{w}_{DO,i} \leftarrow KWCiphertextGen(gp, w, PK_S, PK_{DO,i})$, $uk_{i\to j} \leftarrow UpdateKeyGen(gp, SK_{DO,i}, SK_{DO,j})$, $CT^{w}_{DO,j} \leftarrow CiphertextUpdate(gp, uk_{i\to j}, CT^{w}_{DO,i})$, $rk_{DO\to AU} \leftarrow ReKeyGen(gp, SK_{DO,j}, SK_{AU})$, $CT^{w}_{AU} \leftarrow CiphertextShare(gp, CT^{w}_{DO,j}, rk_{DO\to AU})$, $T_w \leftarrow SearchTokenGen(gp, w, PK_S, SK_{AU})$, then $1 \leftarrow Test(gp, CT^{w}_{AU}, T_w, SK_S)$.

3.2 Security definitions

A KU-CS-PEKS scheme should meet the indistinguishability of keyword ciphertext/search token. To formalize the security definitions of KU-CS-PEKS, four adversarial games (Game 1 ~ Game 4) between an adversary A_i (i = 1, 2, 3, 4) and a challenger B are defined, where the adversary A_i is either a malicious designated storage server or an outsider attacker. Game 1 defines the original keyword ciphertext indistinguishability under the adaptively chosen keyword attack (OKC-IND-CKA). Game 2 defines the re-encryption keyword ciphertext indistinguishability under the adaptively chosen keyword attack (RKC-IND-CKA). Game 3 defines the data owner's search token indistinguishability under the adaptively chosen keyword attack (DOST-IND-CKA). Game 4 defines the authorized user's search token indistinguishability under the adaptively chosen keyword attack (AUST-IND-CKA).

Game 1 (OKC-IND-CKA): In this game, B plays with A_1 who models a designated storage server or an outsider attacker as below:

Setup: B generates gp and (PK_S, SK_S) by running $GlobalSetup(\lambda)$ and $KeyGen_{server}(gp)$. Then, B gives A_1 {gp, PK_S} if it is an outside attacker or {gp, PK_S, SK_S} if it is a designated storage server.

Phase 1: A_1 is capable of making six oracle queries adaptively.

- Uncorrupted key generation oracle O_{pk}: With the input of a time period index i by the adversary, the challenger runs $KeyGen_{user}(gp)$ to produce a key pair ($PK_{DO,i}$, $SK_{DO,i}$) of the data owner and then gives $PK_{DO,i}$ to A_1.

- Corrupted key generation oracle O_{sk}: With the input of a time period index j by the adversary, the challenger runs $KeyGen_{user}(gp)$ to produce a key pair ($PK_{DO,j}$, $SK_{DO,j}$) of the data owner and then gives ($PK_{DO,j}$, $SK_{DO,j}$) to A_1. Here, it is restricted that A_1 is disallowed to submit a same time period index to both O_{pk} and O_{sk}.

- Update key generation oracle O_{uk}: With the input of two distinct time period indices (k, l) by the adversary, the challenger runs $UpdateKeyGen(gp, SK_{DO,k}, SK_{DO,l})$ to produce an update key $uk_{k\rightarrow l}$ and then returns it to A_1. As in [35], it is not allowed the update key queries to occur on a corrupted time period and an uncorrupted time period.

- Update ciphertext generation oracle O_{uc}: With the input of two distinct time period indices (k, l) and an keyword ciphertext $CT^{w}_{DO,k}$ by the adversary where either both k and l are corrupted or both are uncorrupted, the challenger runs $CiphertextUpdate(gp, uk_{k\rightarrow l}, CT^{w}_{DO,k})$ to produce an update ciphertext $CT^{w}_{DO,l}$ and then returns it to A_1.

- Search token generation oracle O_{st}: With the input of a time period index i and a keyword w by the adversary, the challenger runs $SearchTokenGen(gp, w, PK_S, SK_{DO,i})$ to produce a search token T_w and outputs to A_1.

- Test oracle O_{te}: With the input of $(CT^{w}_{DO,i}, T_{w'})$ by the adversary, the challenger runs $Test(gp, CT^{w}_{DO,i}, T_{w'}, SK_S,)$ and then returns the result to A_1. This oracle only allows the outside attacker to make queries.

Challenge: Once the Phase 1 is finished, A_1 inputs two distinct keywords (w_0, w_1) where $|w_0| = |w_1|$ and a time period index i^* to B. The restrictions are (1) A_1 has never submitted the query $O_{sk}(i^*)$; (2) A_1 has never submitted the queries $O_{st}(i^*, w_0)$ and $O_{st}(i^*, w_1)$ if it is the designated data storage server; (3) For any time period j, A_1 has never submitted the queries $O_{uk}(i^*, j)$ and $O_{st}(j, w_0)$ or the queries $O_{uk}(i^*, j)$ and $O_{st}(j, w_1)$ if it is the designated data storage server. The challenger B randomly picks $b \in \{0, 1\}$, calculates the ciphertext $CT^{w_b}_{DO,i^*} = KWCiphertextGen(gp, w_b, PK_{DO,i^*}, PK_S)$ and sends it to A_1.

Phase 2: A_1 makes more oracle queries, but with the following constraints: (1) A_1 is disallowed to ask $O_{sk}(i^*)$; (2) A_1 is disallowed to ask $O_{te}(CT_{DO,i^*}^{w_b}, T_1^*)$ if it is the outside attacker, where T_1^* is the result of the query $O_{st}(i^*, w_0)$ or $O_{st}(i^*, w_1)$; (3) For any time period j, A_1 is disallowed to ask $O_{te}(O_{uc}(i^*, j, CT_{DO,i^*}^{w_b}), T_2^*)$ if it is the outside attacker, where T_2^* is the result of the query $O_{st}(j, w_0)$ or $O_{st}(j, w_1)$; (4) A_1 is disallowed to ask $O_{st}(i^*, w_b)$ if it is the designated data storage server; (5) For any time period j, A_1 is disallowed to ask both $O_{uk}(i^*, j)$ and $O_{st}(j, w_0)$ or both $O_{uk}(i^*, j)$ and $O_{st}(j, w_1)$ if it is the designated data storage server; (6) For any time period j, A_1 is disallowed to ask both $O_{uc}(i^*, j, CT_{DO,i^*}^{w_b})$ and $O_{st}(j, w_0)$ or both $O_{uc}(i^*, j, CT_{DO,i^*}^{w_b})$ and $O_{st}(j, w_1)$ if A_1 is the designated data storage server.

Guess: At last, A_1 outputs a guess b' and wins if $b = b'$. Its advantage is defined as $Adv_{A_1}^{OKC-IND-CKA}(\lambda) = |\Pr[b = b'] - 1/2|$.

Definition 4. For a KU-CS-PEKS scheme, if the advantage $Adv_{A_1}^{OKC-IND-CKA}$ of any polynomial time adversary A_1 is negligible, then the scheme is OKC-IND-CKA secure.

Game 2 (RKC-IND-CKA): In this game, B plays with A_2 who models a designated storage server or an outsider attacker as below:

Setup: B generates gp, (PK_S, SK_S) and ($PK_{DO,i}$, $SK_{DO,i}$) by running $GlobalSetup(\lambda)$, $KeyGen_{server}(gp)$ and $KeyGen_{user}(gp)$ respectively. Then, B gives A_2 $\{gp, PK_S, PK_{DO,i}\}$ if it is an outside attacker or $\{gp, PK_S, SK_S, PK_{DO,i}\}$ if it is a designated storage server.

Phase 1: A_2 is capable of making six oracle queries adaptively.

- Uncorrupted key generation oracle O_{pk}: When A_2 queries this oracle, the challenger runs $KeyGen_{user}(gp)$ to produce an authorized user's public/secret key pair (PK_{AU}, SK_{AU}) and gives PK_{AU} to A_2.
- Corrupted key generation oracle O_{sk}: When A_2 queries this oracle, the challenger runs $KeyGen_{user}(gp)$ to produce an authorized user's public/secret key pair (PK_{AU}, SK_{AU}) and gives (PK_{AU}, SK_{AU}) to A_2.
- Re-encryption key generation oracle O_{rk}: With the input of ($PK_{DO,i}$, PK_{AU}) by the adversary, the challenger runs $ReKeyGen(gp, SK_{DO,i}, SK_{AU})$ to produce a re-encryption key $rk_{DO \to AU}$ and then returns it to A_2.
- Share ciphertext generation oracle O_{sc}: With the input of ($CT_{DO,i}^{w}$, $rk_{DO \to AU}$) by the adversary, the challenger runs $CiphertextShare(gp, CT_{DO,i}^{w}, rk_{DO \to AU})$ to produce a share ciphertext CT_{AU}^{w} and then returns it to A_2.
- Search token generation oracle O_{st}: With the input of (PK_{AU}, w) by the adversary, the challenger runs $SearchTokenGen(gp, w, PK_S, SK_{AU})$ to produce a search token T_w and then returns it to A_2.
- Test oracle O_{te}: With the input of (CT_{AU}^{w}, T_w) by the adversary, the challenger runs $Test(gp, CT_{AU}^{w}, T_w, SK_S)$ and returns the result to A_2.

Challenge: Once the Phase1 is finished, A_2 inputs two distinct keywords (w_0, w_1) where $|w_0| = |w_1|$ and a public key PK_{AU}^*. The restrictions are (1) PK_{AU}^* is from O_{pk}; (2) A_2 has never submitted the queries $O_{st}(PK_{AU}^*, w_0)$ and $O_{st}(PK_{AU}^*, w_1)$. B randomly picks $b \in \{0, 1\}$, calculates the ciphertext $C^* = KWCiphertextGen(gp, w_b, PK_S, PK_{AU}^*)$ and sends it to A_2.

Phase 2: A_2 makes more oracle queries. The only restriction is that A_2 has never submitted the queries $O_{st}(PK_{AU}^*, w_0)$ and $O_{st}(PK_{AU}^*, w_1)$.

Guess: At last, A_2 outputs a guess b' and wins if $b = b'$. A_2's advantage is defined to be $Adv_{A_2}^{RKC-IND-CKA}(\lambda) = |\Pr[b = b'] - 1/2|$.

Definition 5. For a KU-CS-PEKS scheme, if the advantage $Adv_{A_2}^{RKC-IND-CKA}$ of any polynomial time adversary A_2 is negligible, then the scheme is RKC-IND-CKA secure.

Game 3 (DOST-IND-CKA): In this game, B plays with A_3 who models an outside attacker as below:

Setup: B generates gp and (PK_S, SK_S) by running $GlobalSetup(\lambda)$ and $KeyGen_{server}(gp)$ respectively. Then, B gives $A_3\{gp, PK_S\}$.

Phase 1: A_3 is capable of make queries to O_{pk}, O_{sk}, O_{uk}, O_{uc}, O_{st} and O_{te} adaptively. These queries are answered as in OKC-IND-CKA-Game.

Challenge: Once the Phase1 is finished, A_3 inputs two distinct keywords (w_0, w_1) where $|w_0| = |w_1|$ and a time period index i^* to B. The restrictions are (1) A_3 has never submitted the query $O_{sk}(i^*)$; (2) For any time period j, A_3 has never submitted the queries $O_{uk}(j, i^*)$; (3) For any time period j, A_3 has never submitted the queries $O_{uc}(j, i^*, CT_{DO,j}^{w_0})$ and $O_{uc}(j, i^*, CT_{DO,j}^{w_1})$. B randomly selects $b \in \{0, 1\}$, calculates a search token $T_{w_b} = SearchTokenGen(gp, w_b, PK_S, SK_{DO,i^*})$ and sends it to A_3.

Phase 2: A_3 makes more oracle queries, but with the constraints: (1) A_3 is disallowed to ask $O_{sk}(i^*)$; (2) For any time period j, A_3 is disallowed to ask $O_{uk}(j, i^*)$; (3) For any time period j, A_3 is disallowed to ask $O_{uc}(j, i^*, CT_{DO,j}^{w_0})$ and $O_{uc}(j, i^*, CT_{DO,j}^{w_1})$.

Guess: At last, A_3 outputs a guess b' and wins if $b = b'$. Its advantage is defined to be $Adv_{A_3}^{DOST-IND-CKA}(\lambda) = |\Pr[b = b'] - 1/2|$.

Definition 6. For a KU-CS-PEKS scheme, if the advantage $Adv_{A_3}^{DOST-IND-CKA}$ of any polynomial time adversary A_3 is negligible, then the scheme is DOST-IND-CKA secure.

Game 4 (AUST-IND-CKA): In this game, B plays with A_4 who models an outside attacker as below:

Setup: B generates gp, (PK_S, SK_S) and ($PK_{DO,i}$, $SK_{DO,i}$) by running $GlobalSetup(\lambda)$, $KeyGen_{server}(gp)$ and $KeyGen_{user}(gp)$ respectively. Then, B gives A_4 $\{gp, PK_S, PK_{DO,i}\}$.

Phase 1: A_4 is capable of make queries to O_{pk}, O_{sk}, O_{rk}, O_{sc}, O_{st} and O_{te} adaptively. These queries are answered as in RKC-IND-CKA-Game.

Challenge: Once the Phase1 is finished, A_4 inputs two distinct keywords (w_0, w_1) where $|w_0| = |w_1|$ and a public key PK_{AU}^*. The restrictions are that: (1) PK_{AU}^* is from O_{pk}; (2) A_4 has never submitted the queries $O_{rk}(PK_{DO,i}, PK_{AU}^*)$; (3) A_4 has never submitted the queries $O_{sc}(CT_{DO,i}^{w_0}, rk_{DO \to AU^*})$ and $O_{sc}(CT_{DO,i}^{w_1}, rk_{DO \to AU^*})$. B randomly selects $b \in \{0, 1\}$, calculates a search token $T_{w_b} = SearchTokenGen(gp, w_b, PK_S, SK_{AU}^*)$ and sends it to A_4.

Phase 2: A_4 makes more oracle queries, but with the constraints: (1) A_4 has never submitted the queries $O_{rk}(PK_{DO,i}, PK_{AU}^*)$; (2) A_4 has never submitted the queries $O_{sc}(CT_{DO,i}^{w_0}, rk_{DO \to AU^*})$ and $O_{sc}(CT_{DO,i}^{w_1}, rk_{DO \to AU^*})$.

Guess: At last, A_4 outputs a guess b' and wins if $b = b'$. Its advantage is defined as $Adv_{A_4}^{AUST-IND-CKA}(\lambda) = |\Pr[b = b'] - 1/2|$.

Definition 7. For a KU-CS-PEKS scheme, if the advantage $Adv_{A_4}^{AUST-IND-CKA}$ of any polynomial time adversary A_4 is negligible, then the scheme is AUST-IND-CKA secure.

4. The proposed KU-CS-PEKS scheme

4.1 Description of the proposed scheme

The proposed KU-CS-PEKS scheme is as below:

(1) *GlobalSetup*(λ): Inputting λ, the algorithm generates two cyclic groups (G, G_T) with order p and a bilinear pairing e: $G \times G \rightarrow G_T$. Additionally, it selects a generator $g \in G$ and three cryptographic hash functions H: $G \rightarrow \{0, 1\}^{hlen}$, H_1: $\{0, 1\}^* \rightarrow G$ and H_2: $G_T \rightarrow \{0, 1\}^{hlen}$. Finally, it sets $gp = \{p, g, G, G_T, e, H, H_1, H_2\}$.

(2) $KeyGen_{server}(gp)$: Inputting gp, this algorithm picks $a \in Z_p^*$ at random, sets $SK_S = a$, and calculates $PK_S = g^a$. It then outputs (PK_S, SK_S).

(3) $KeyGen_{user}(gp)$: Inputting gp, the algorithm picks $x_U \in Z_p^*$ at random, sets $SK_U = x_U$ and $PK_U = g^{x_U}$. It then outputs (PK_U, SK_U). In the following algorithms, the data owner's key pairs in different time periods i and j are denoted by ($PK_{DO,i}$, $SK_{DO,i}$) and ($PK_{DO,j}$, $SK_{DO,j}$) respectively, and the authorized user's key pair is denoted by (PK_{AU}, SK_{AU}).

(4) *KWCiphertextGen*(gp, w, PK_S, $PK_{DO,i}$): Inputting gp, w, PK_S and $PK_{DO,i}$, this algorithm randomly picks $r \in Z_p^*$, calculates $A = PK_{DO,i}^r$ and $B = H_2(t)$, where $t = e(PK_S, H_1(w))^r$. It then outputs $CT_{DO,i}^w = (A, B)$.

(5) *UpdateKeyGen*(gp, $SK_{DO,i}$, $SK_{DO,j}$): Inputting gp, $SK_{DO,i}$ and $SK_{DO,j}$, the data owner calculates a update key $uk_{i\rightarrow j} = SK_{DO,j} / SK_{DO,i} \bmod p$.

(6) *CiphertextUpdate*(gp, $uk_{i\rightarrow j}$, $CT_{DO,i}^w$): Inputting gp, an update key $uk_{i\rightarrow j}$ and a keyword ciphertext $CT_{DO,i}^w = (A, B)$ of the time period i. The algorithm calculates $A' = A^{uk_{i\rightarrow j}} = g^{SK_{DO,j} \cdot r} = PK_{DO,j}^r$ and creates a new keyword ciphertext $CT_{DO,j}^w = (A', B)$ for the time period j.

(7) *ReKeyGen*(gp, $SK_{DO,j}$, SK_{AU}): Inputting gp, $SK_{DO,j}$ and SK_{AU}, this algorithm generates a re-encryption key $rk_{DO\rightarrow AU} = SK_{AU} / SK_{DO,j} \bmod p$ as below: The data owner picks $n \in Z_p^*$ at random and sends ($nSK_{DO,j} \bmod p$) to the authorized user; The authorized user calculates and sends $SK_{AU} / nSK_{DO,j} \bmod p$ to the data owner; Finally, the data owner uses can calculate $rk_{DO\rightarrow AU} = SK_{AU} / SK_{DO,j} \bmod p$.

(8) *CiphertextShare*(gp, $CT_{DO,j}^w$, $rk_{DO\rightarrow AU}$): Inputting gp, $CT_{DO,j}^w$ and $rk_{DO\rightarrow AU}$, the algorithm calculates $A'' = A^{rk_{DO\rightarrow AU}} = g^{SK_{AU}} = PK_{AU}^r$ and creates a shared keyword ciphertext $CT_{AU}^w = (A'', B)$ for the authorized user.

(9) *SearchTokenGen*(gp, w', PK_S, SK_U): Inputting gp, w', PK_S and SK_U, the algorithm picks $r' \in Z_p^*$ at random, calculates $T_1 = g^{r'}$ and $T_2 = H_1(w')^{\frac{1}{SK_U}} \cdot H(PK_S^{r'})$. It then outputs $T_{w'} = (T_1, T_2)$.

(10) *Test*(gp, CT_w, T_w, SK_S): Inputting gp, CT_w, $T_{w'}$ and $SK_S = a$,, this algorithm first computes $\tau = T_2 / H(T_1^a)$ and then tests if $B = H_2(e(A, \tau^a))$. If yes, it returns 1, else, it returns 0.

Correctness: Assume the ciphertext of keyword w is $CT_w = (PK_U^r, H_2(e(g^a, H_1(w))^r))$ and the search token associated to keyword w' is $T_{w'} = (g^{r'}, H_1(w')^{\frac{1}{SK_U}} \cdot H(PK_S^{r'}))$. If $w = w'$, then

$$B = H_2(e(g^a, H_1(w))^r) = H_2(e(g^{x'r}, (H_1(w')^{\frac{1}{x'}})^a)) = H_2(e(PK_U^r, (T_2 / H(T_1^a))^a)) = H_2(e(A, \tau^a)).$$

Therefore, the KU-CS-PEKS scheme is correct.

4.2 Security proofs

Theorem 1. The KU-CS-PEKS scheme satisfies the OKC-IND-CKA security under the hardness assumption of the 1-BDHI problem in the random oracle model.

Proof: Presuming that an adversary A_1 can break the OKC-IND-CKA security of the KU-CS-PEKS scheme with an advantage ε. Then an algorithm B can be created to solve the 1-BDHI problem with advantage $\varepsilon' = \varepsilon / eq_{st}q_{H_2}$, where q_{H_2} and q_{st} respectively represent the largest number of A_1's queries to the oracles O_{H_2} and O_{st}, and e denotes the base of the natural logarithm. Given an instance $\{p, g, G, G_T, e, u_1 = g^y\}$ of the 1-BDHI problem, B's goal is to calculate $e(g,g)^{1/y} \in G$.

B plays with A_1 as below:

Setup: B picks $a \in Z_p^*$ at ransom, sets $PK_S = g^a$ and $SK_S = a$. Then, B gives $gp = \{p, g, G, G_T, e, H, H_1, H_2\}$ and PK_S to A_1 if it is an outside attacker or $gp = \{p, g, G, G_T, e, H, H_1, H_2\}$ and (PK_S, SK_S) if it is a designated data storage server, where H, H_1 and H_2 are three random oracles.

Phase 1: A_1 is capable of make these queries:

- Random oracle O_H : B holds a list H-list that is made up of the tuples <M, N>. When getting a query $H(M)$ from A_1, B outputs N if <M, N> is already in H-list. Otherwise, B randomly picks $N \in \{0, 1\}^{hlen}$ and sets $H(M) = N$. Finally, B records a new tuple <M, N> onto H-list and responds with N.

- Random oracle O_{H_1} : B holds a list H_1-list that is made up of tuples <w, h, e, $cion$>. When getting a query $H_1(w)$ from A_1, B outputs h if <w, h, e, $cion$> is already in the H_1-list. Else, B randomly picks $cion \in \{0, 1\}$ such that Pr[$cion$ = 0] =1 / (q_{st} + 1). Then, B randomly picks $e \in Z_p^*$ and calculates $h = g^e \in G$ if $cion$ = 0 or $h = (u_1)^e = g^{ye} \in G$ otherwise. Finally, B records a new tuple <w, h, e, $cion$> onto H_1-list and responds with $H_1(w) = h$.

- Random oracle O_{H_2} : B holds a list H_2-list that is made up of the tuples <t, V>. When getting a query $H_2(t)$ from A_1, B outputs V if <t, V> is already in H_2-list. Otherwise, B randomly selects $V \in \{0, 1\}^{hlen}$ and sets $H_2(t) = V$. Finally, B records a new tuple <t, V> onto H_2-list and returns V.

- Uncorrupted key generation oracle O_{pk}: B holds a list L_U that is made up of the tuples <i, $PK_{DO,i}$, x_i>. When A_1 inputs a time period index i, B responds as below:

B responds with $PK_{DO,i}$, if i is already in the L_U with a tuple <i, $PK_{DO,i}$, x_i>.

Else, B randomly selects $x_i \in Z_p^*$ and sets $PK_{DO,i} = g^{x_i}$. Then B records the tuple <i, $PK_{DO,i}$, x_i> onto L_U and returns $PK_{DO,i}$.

- Corrupted key generation oracle O_{sk}: B holds a list L_C that is made up of tuples <j, $PK_{DO,j}$, x_j>. When A_1 inputs a time period index j and $j \neq i$, algorithm B responds as below:

If j is already in the L_C with a tuple <j, $PK_{DO,j}$, x_j>, then B answers with ($PK_{DO,j}$, $SK_{DO,j} = x_j$).

Otherwise, B chooses $x_j \in Z_p^*$ at random and sets $PK_{DO,j} = g^{x_j}$. Then B records the tuple <j, $PK_{DO,j}$, x_j> onto L_C and returns ($PK_{DO,j}$, $SK_{DO,j} = x_j$).

- Update key generation oracle O_{uk}: With the input of two distinct time period indices (k, l) by the adversary. It is required that both $PK_{DO,k}$ and $PK_{DO,l}$ appear on the L_U or L_C, B responds

A_1 with $uk_{k\to l} = x_l / x_k$.

- Update ciphertext generation oracle O_{uc}: With the input of two distinct time period indices (k, l) and a keyword ciphertext $CT^w_{DO,k}$ by the adversary, $PK_{DO,k}$ and $PK_{DO,l}$ both appear on the L_U or L_C. B obtains the update key $uk_{k\to l} = x_l / x_k$ from O_{uk} and returns the update ciphertext $CT^w_{DO,l} = (A, B) = (PK^r_{DO,l}, H_2(e(PK_S, H_1(w))^r))$.

- Search token generation oracle O_{st}: With the input of a time period index i and a keyword w by the adversary, algorithm B responds as follows:

B gets from O_{H_1} a tuple <w, h, e, $cion$> from O_{H_1}. If $cion = 0$, B ends the game.

Else, B selects $r' \in Z^*_p$, $y' \in Z^*_p$ at random, sets $T_1 = g^{r'}$ and $T_2 = (u_1^e)^{1/y\cdot y'} \cdot H(PK_S^{r'}) = g^{e/y'} \cdot H(g^{a\cdot r'})$. Because $PK_U = (u_1)^{y'} = g^{x_i}$ and $H_1(w)^{1/x_i} = (g^{y\cdot e})^{1/y\cdot y'} = g^{e/y'}$, $T_w = (T_1, T_2)$ is a valid search token for w. B responds A_1 with $T_w = (T_1, T_2)$.

-Test oracle O_{te}: With the input of $(CT^w_{DO,l}, T_{w'})$ by the adversary, the algorithm B outputs 1 if $B = H_2(e(A, \tau^a)$ holds or 0 otherwise.

Challenge: Once the Phase1 is finish, A_1 inputs two distinct keywords (w_0, w_1) where $|w_0| = |w_1|$ and a time period index i^* to B. B asks O_{H_1} to obtain $h_b \in G$ such that $H_1(w_b) = h_b$, where $b \in \{0, 1\}$. Let <w_b, h_b, e_b, $coin_b$> be the response tuple. If both $coin_0 = 1$ and $coin_1 = 1$, B ends the game. Else, B randomly picks $k' \in Z^*_p$ and lets $r = k' / a \cdot y \in Z^*_p$, and calculates $A^* = PK^r_{DO,i^*} = (g^{y\cdot y'})^r = (g^{y\cdot y'})^{k'/a\cdot y} = g^{y'\cdot k'/a}$; chooses $Z \in \{0, 1\}^{hlen}$ at random and sets $B^* = Z$. Finally, it transmits the challenge ciphertext $C^* = (A^*, B^*)$ to A_1.

Phase 2: A_1 makes more oracle queries, but with the constraints as defined in Game 1.

Guess: At last, A_1 outputs its guess b'. Clearly, A_1 may have issued a query $H_2(e(PK_S^r, H_1(w_b))) = H_2(e(g^a, g^{e_b})^{\frac{k'}{a\cdot x}})$. Therefore, it is possible that there exists one pair of the form $(e(g^a, g)^{\frac{e_b\cdot k'}{x}}, H_2(e(PK_S^r, H_1(w_b))))$ in H_2-list. B picks a tuple <t, V> at random from H_2-list and outputs $t^{1/e_b\cdot k'}$ as its solution to the given 1-BDHI problem.

In order to analyze the advantage of B in solving the given 1-BDHI problem, three events are defined:

δ_1 : B does not terminate for A_1's search token query;

δ_2 : B does not terminate during Challenge Phase;

δ_3 : A_1 does not issue a query for either $H_2(e(PK_S^r, H_1(w_0)))$ or $H_2(e(PK_S^r, H_1(w_1)))$.

Clearly, $\Pr[\delta_1] = 1 - 1/(q_{st} + 1)$, since A_1 makes at most q_{st} search token queries, the probability that B does not abort as result of all search token queries is at least $(1 - 1/(q_{st} + 1))^{q_{st}} \geq 1/e$.

In the challenge phase, $\Pr[c_b = 0] = 1/(q_{st} + 1)$ where $b \in \{0, 1\}$. Therefore, $\Pr[c_0 = c_1 = 1] = (1 - 1/(q_{st} + 1))^2 \leq 1 - 1/q_{st}$. Thus, the probability of event δ_2 is at least $1/q_{st}$.

Assuming that A_1 does not query either $H_2(e(PK_S^r, H_1(w_0)))$ or $H_2(e(PK_S^r, H_1(w_1)))$, then $\Pr[b = b' \mid \neg\delta_3] = 1/2$. According to the total probability formula:

$$\Pr[b = b'] = \Pr[b = b' \mid \delta_3]\Pr[\delta_3] + \Pr[b = b' \mid \neg\delta_3]\Pr[\neg\delta_3]$$
$$\leq \Pr[b = b' \mid \delta_3]\Pr[\delta_3] + \Pr[\neg\delta_3]$$

$$= \frac{1}{2} + \frac{1}{2}\Pr[\neg\delta_3].$$

Therefore, $\varepsilon \le |\Pr[b = b'] - \frac{1}{2}| \le \frac{1}{2}\Pr[\neg\delta_3]$. Consequently, $\Pr[\neg\delta_3] \ge 2\varepsilon$

Since the probability that B chooses the right tuple from H_2-list is at least $1/q_{H_2}$, B's success probability is at least $\varepsilon / (eq_{st}q_{H_2})$.

Theorem 2. The KU-CS-PEKS scheme satisfies the RKC-IND-CKA security under the hardness assumption of the 1-BDHI problem in the random oracle model.

Proof: Presuming that an adversary A_2 can break the OKC-IND-CKA security of the KU-CS-PEKS scheme with an advantage ε. Then an algorithm B can be created to solve the 1-BDHI problem with advantage $\varepsilon' = \varepsilon / eq_{st}q_{H_2}$, where q_{H_2} and q_{st} respectively denote the largest number of A_2's queries to the oracles O_{H_2} and O_{st}, and e is the base of the natural logarithm. Given an instance $\{p, g, G, G_T, e, u_1 = g^y\}$ of the 1-BDHI problem, B's goal is to calculate $e(g,g)^{1/y} \in G$.

B plays with A_2 as below:

Setup: B picks $a \in Z_p^*$ at random, sets $PK_S = g^a$ and $SK_S = a$. Then, B gives $gp = \{p, g, G, G_T, e, H, H_1, H_2\}$ and PK_S to A_2 if it is an outside attacker or or $gp = \{p, g, G, G_T, e, H, H_1, H_2\}$ and (PK_S, SK_S) if it is a designated data storage server, where H, H_1 and H_2 are three random oracles.

Phase 1: A_2 is capable of make these queries:

- Random oracle O_H : B holds a list H-list that is made up of the tuples <M, N>. When getting a query $H(M)$ from A_2, B outputs N if <M, N> is already in H-list. Otherwise, B randomly picks $N \in \{0, 1\}^{hlen}$ and sets $H(M) = N$. Finally, B records a new tuple <M, N> onto H-list and responds with N.

- Random oracle O_{H_1} : B holds a list H_1-list that is made up of tuples <w, h, e, $cion$>. When getting a query $H_1(w)$ from A_2, B outputs h if <w, h, e, $cion$> is already in H_1-list. Else, B randomly picks $cion \in \{0, 1\}$ such that $\Pr[cion = 0] = 1 / (q_{st} + 1)$. Then, B randomly picks e and calculates $h = g^e \in G$ if $cion = 0$ or $h = (u_1)^e = g^{ye} \in G$ otherwise. Finally, B records a new tuple <w, h, e, $cion$> onto H_1-list and responds with $H_1(w) = h$.

- Random oracle O_{H_2} : B holds a list H_2-list that is made up of the tuples <t, V>. When getting a query $H_2(t)$ from A_2, B outputs V if <t, V> is already in H_2-list. Otherwise, B randomly selects $V \in \{0, 1\}^{hlen}$ and sets $H_2(t) = V$. Finally, B records a new tuple <t, V> onto H_2-list and returns V.

- Uncorrupted key generation oracle O_{pk}: B holds a list L_U that is made up of the tuples <PK_{AU}, x_{AU}>. When A_2 queries this oracle, B selects $x_{AU} \in Z_p^*$ at random and sets $PK_{AU} = g^{x_{AU}}$. B records the tuple <PK_{AU}, x_{AU}> onto L_U and returns PK_{AU}.

- Corrupted key generation oracle O_{sk}: B holds a list L_C that is made up of the tuples <PK_{AU}, x_{AU}>. When A_2 queries this oracle, B selects $x_{AU} \in Z_p^*$ at random and sets $PK_{AU} = g^{x_{AU}}$. B records the tuple <PK_{AU}, x_{AU}> onto L_C and returns ($PK_{AU} = g^{x_{AU}}$, $SK_{AU} = x_{AU}$).

- Re-encryption key generation oracle O_{rk}: With the input of ($PK_{DO,i}$, PK_{AU}) by the adversary, B returns the re-encryption key $rk_{DO \to AU} = x_{AU} / x_i$.

- Share ciphertext generation oracle O_{sc}: With the input of ($CT^{w}_{DO,i}$, $rk_{DO\to AU}$) by the adversary, B returns the share ciphertext $CT^{w}_{AU} = (A, B) = (PK^{r}_{AU} , H_2(e(PK_S, H_1(w))^{r}))$.

- Search token generation oracle O_{st}: With the input of (w, PK_{AU}) by the adversary, B responds as below:

B gets from O_{H_1} a tuple <w, h, e, $cion$> from O_{H_1}. If $cion$ = 0, B ends the game.

Else, $h = (u_1)^{e}$. B selects $r' \in Z_p^*$, $y' \in Z_p^*$ at random, sets $T_1 = g^{r'}$ and $T_2 = (u_1)^{1/y\cdot y'} \cdot H(PK_S^{r'}) = g^{e/y'} \cdot H(g^{a\cdot r'})$. Because $PK_U = (u_1)^{y'} = g^{x_i}$ and $H_1(w)^{1/x_i} = (g^{y\cdot e})^{1/y\cdot y'} = g^{e/y'}$, $T_w = (T_1, T_2)$ is a valid search token for w. B responds A_2 with $T_w = (T_1, T_2)$.

-Test oracle O_{te}: With the input of ($CT^{w}_{DO,l}$, $T_{w'}$) by the adversary, B outputs 1 if $B = H_2(e(A, \tau^{a})$ holds or 0 otherwise.

Challenge: Once the Phase1 is finished, A_2 inputs two distinct keywords (w_0, w_1) where $|w_0| = |w_1|$ and a time period index i^* to B. B asks O_{H_1} to obtain h_b such that $H_1(w_b) = h_b$, where $b \in \{0, 1\}$. Let <w_b, h_b, e_b, $coin_b$> be the response tuple. If both $coin_0 = 1$ and $coin_1 = 1$, B ends the game. Else, B randomly picks $k' \in Z_p^*$ and lets $r = k'/a\cdot y \in Z_p^*$, and calculates $A^* = PK^{r}_{DO,i*} = (g^{y\cdot y'})^{r} = (g^{y\cdot y'})^{k'/a\cdot y} = g^{y'\cdot k'/a}$; chooses $Z \in \{0, 1\}^{hlen}$ at random and sets $B^* = Z$. Finally, it responds with the challenge ciphertext $C^* = (A^*, B^*)$.

Phase 2: A_2 makes more oracle queries, but with the constraints as defined in Game 2.

Guess: At last, A_2 outputs a guess $b' \in \{0, 1\}$. B randomly picks a tuple (t, V) from H_2-list and outputs $t^{1/e_b\cdot k'}$ as its solution to the given 1-BDHI problem.

Similar to the proof of Theorem 1, it can be deduced that the lower bound on B's advantage is $\varepsilon / (eq_{td}q_{H_2})$.

Theorem 3. The KU-CS-PEKS scheme satisfies the DOST-IND-CKA security under the hardness assumption of the HDH problem in the random oracle model.

Proof: Presuming that an adversary A_3 breaks the DOST -IND-CKA security of the KU-CS-PEKS scheme with an advantage ε. Then an algorithm B can be created to solve the HDH problem with advantage $\varepsilon' = \varepsilon$. Given an instance {$p$, g, G, G_T, e, g^a, g^b, η, H} of the HDH problem, where H: $G \to \{0, 1\}^{hlen}$ is a hash function and η is either $H(g^{ab})$ or a random element of G. B's goal is to decide whether $\eta = H(g^{ab})$.

B plays with A_3 as below:

Setup: B picks a, $l \in Z_p^*$ at random, sets $PK_S = g^{al}$ and $SK_S = al$. Then, B gives gp = {p, g, G, G_T, e, H, H_1, H_2} and PK_S to A_3, where H, H_1 and H_2 are three random oracles.

Phase 1: A_3 is capable of make these queries:

- Random oracle O_H : B holds a list H-list that is made up of the tuples <M, N>. When getting a query $H(M)$ from A_4, B outputs N if <M, N> is already in H-list. Otherwise, B randomly picks $N \in \{0, 1\}^{hlen}$ and sets $H(M) = N$. Finally, B records a new tuple <M, N> onto H-list and responds with N.

- Random oracle O_{H_1} : B holds a list H_1-list that is made up of tuples <w, h>. When getting a query $H_1(w)$ from A_4, B outputs h if <w, h> is already in H_1-list. Otherwise, B randomly picks $h \in G$ and sets $H_1(w) = h$. Finally, B records a new tuple <w, h> onto H_1-list and returns h.

- Random oracle O_{H_2} : B holds a list H_2-list that is made up of the tuples <t, V>. When getting a query $H_2(t)$ from A_4, B outputs V if <t, V> is already in H_2-list. Otherwise, B randomly selects $V \in \{0, 1\}^{hlen}$ and sets $H_2(t) = V$. Finally, B records a new tuple <t, V> onto H_2-list and

returns V.

- Uncorrupted key generation oracle O_{pk}: B holds a list L_U that is made up of the tuples $<i, PK_{DO,i}, x_i>$. When A_3 inputs a time period index i, B responds as below: B responds with $PK_{DO,i}$, if i is already in L_U with a tuple $<i, PK_{DO,i}, x_i>$. Else, B randomly selects $x_i \in Z_p^*$ and sets $PK_{DO,i} = g^{x_i}$. Then B records the tuple $<i, PK_{DO,i}, x_i>$ onto L_U and returns $PK_{DO,i}$.

- Corrupted key generation oracle O_{sk}: B holds a list L_U of tuples $<j, PK_{DO,j}, x_j>$. When A_3 inputs a time period index j and $j \neq i$, B responds as below: If j is already in L_C with a tuple $<j, PK_{DO,j}, x_j>$, then B answers with ($PK_{DO,j} = g^{x_j}$, $SK_{DO,j} = x_j$). Otherwise, B selects a random value x_j and sets $PK_{DO,j} = g^{x_j}$. Then B records the tuple $<j, PK_{DO,j}, x_j>$ onto the L_C and returns ($PK_{DO,j} = g^{x_j}$, $SK_{DO,j} = x_j$).

- Update key generation oracle O_{uk}: With the input of two distinct time period indices (k, l) by the adversary, $PK_{DO,k}$ and $PK_{DO,l}$ both appear on the L_U or L_C. B responds A_3 with $uk_{k\to l} = x_l / x_k$.

- Update ciphertext generation oracle O_{uc}: With the input of two distinct time period indices (k, l) and a keyword ciphertext $CT_{DO,k}^{w}$ by the adversary. $PK_{DO,k}$ and $PK_{DO,l}$ both appear on the L_U or L_C. B obtains the update key $uk_{k\to l} = x_l / x_k$ from O_{uk} and returns the update ciphertext $CT_{DO,l}^{w} = (A, B) = (PK_{DO,l}^{r}, H_2(e(PK_S, H_1(w))^r))$.

- Search token generation oracle O_{st}: With the input of a time period index i and a keyword w by A_3, B picks $r' \in Z_p^*$ at random and calculates $T_1 = g^{r'}$ and $T_2 = H_1(w')^{1/x_i} \cdot H(PK_S^{r'})$. B returns A_3 with $T_w = (T_1, T_2)$.

-Test oracle O_{te}: With the input of $(CT_{DO,i}^{w}, T_{w'})$ by the adversary, the algorithm B outputs 1 if the equation $B = H_2(e(A, \tau^a))$ holds or 0 otherwise.

Challenge: Once the Phase1 is finish, A_3 inputs two distinct keywords (w_0, w_1) where $|w_0| = |w_1|$ and a time period index i^*. B picks a value $c \in \{0, 1\}$ at random and sets $T_1^* = g^{b/l}$ and $T_2^* = H_1(w_c)^{1/x_{i^*}} \cdot \eta$. If $\eta = H(g^{ab})$, then $T_{w_c}^*$ is a valid challenge token. If $r^* = b / l$, then $T_1^* = g^{r^*}$, $T_2^* = H_1(w_c)^{1/x_{i^*}} \cdot \eta = H_1(w_c)^{1/SK_{DO,i^*}} \cdot H(g^{ab}) = H_1(w_c)^{1/SK_{DO,i^*}} \cdot H(g^{al\cdot(b/l)}) = H_1(w_c)^{1/SK_{DO,i^*}} H(PK_S^{r'})$. Finally, it responds with the challenge token $T_{w_c}^* = (T_1^*, T_2^*)$.

Phase 2: A_3 makes more oracle queries, but with the constraints as defined in Game 3.

Guess: At last, A_3 outputs a guess c'. If $c = c'$, B output 1, meaning that $\eta = H(g^{ab})$or 0 otherwise.

The advantage of B in solving the given HDH problem is analyzed as below.

According to Game 3, when $\eta = H(g^{ab})$, the view of the adversary A_3 in common with its guess c' satisfies $|\Pr[c = c'] - 1/2| = \varepsilon$. On the other side, when η' is uniform over G, its guess c' satisfies $\Pr[c = c'] = 1/2$. Consequently, the advantage of B in solving the given HDH problem is $|\Pr[B(g, g^a, g^b, H(g^{ab})) = 0] - \Pr[B(g, g^a, g^b, \eta')) = 0]| \geqq |1/2 \pm \varepsilon - 1/2| = \varepsilon$.

Theorem 4. The KU-CS-PEKS scheme satisfies the AUST-IND-CKA security under the hardness assumption of the HDH problem in the random oracle model.

Proof: Presuming that an adversary A_4 breaks the AUST -IND-CKA security of the KU-CS-PEKS scheme with an advantage ε. Then an algorithm B can be created to solve the HDH problem with advantage $\varepsilon' = \varepsilon$. Given an instance $\{p, g, G, G_T, e, g^a, g^b, \eta, H\}$ of the HDH problem, where H: $G \to \{0, 1\}^{hlen}$ is a hash function and η is either $H(g^{ab})$ or a random element

of G. B's goal is to decide whether $\eta = H(g^{ab})$. B plays with A_4 as below:

Setup: B randomly picks $a, l \in Z_p^*$, sets $PK_S = g^{al}$ and $SK_S = al$. Then, B gives $gp = \{p, g, G, G_T, e, H, H_1, H_2\}$ and PK_S to A_4, where H, H_1 and H_2 are three random oracles.

Phase 1: A_4 is capable of make these queries:

- Random oracle O_H : B holds a list H-list that is made up of the tuples <M, N>. When getting a query $H(M)$ from A_4, B outputs N if <M, N> is already in H-list. Otherwise, B randomly picks $N \in \{0, 1\}^{hlen}$ and sets $H(M) = N$. Finally, B records a new tuple <M, N> onto H-list and responds with N.

- Random oracle O_{H_1} : B holds a list H_1-list that is made up of tuples <w, h>. When getting a query $H_1(w)$ from A_4, B outputs h if <w, h> is already in H_1-list. Otherwise, B randomly picks $h \in G$ and sets $H_1(w) = h$. Finally, B records a new tuple <w, h> onto H_1-list and returns h.

- Random oracle O_{H_2} : B holds a list H_2-list that is made up of the tuples <t, V>. When getting a query $H_2(t)$ from A_4, B outputs V if <t, V> is already in H_2-list. Otherwise, B randomly selects $V \in \{0, 1\}^{hlen}$ and sets $H_2(t) = V$. Finally, B records a new tuple <t, V> onto H_2-list and returns V.

- Uncorrupted key generation oracle O_{pk}: B holds a list L_U that is made up of the tuples <PK_{AU}, x_{AU}>. When A_4 queries this oracle, B selects $x_{AU} \in Z_p^*$ at random and sets $PK_{AU} = g^{x_{AU}}$. B records the tuple <PK_{AU}, x_{AU}> onto L_U and returns PK_{AU}.

- Corrupted key generation oracle O_{sk}: B holds a list L_C that is made up of the tuples <PK_{AU}, x_{AU}>. When A_2 queries this oracle, B selects $x_{AU} \in Z_p^*$ at random and sets $PK_{AU} = g^{x_{AU}}$. B records the tuple <PK_{AU}, x_{AU}> onto L_C and returns ($PK_{AU} = g^{x_{AU}}$, $SK_{AU} = x_{AU}$).

- Re-encryption key generation oracle O_{rk}: With the input of ($PK_{DO,i}$, PK_{AU}) by the adversary, algorithm B returns the re-encryption key $rk_{DO \to AU} = x_{AU} / x_i$.

- Share ciphertext generation oracle O_{sc}: With the input of ($CT_{DO,i}^{w}$, $rk_{DO \to AU}$) by the adversary, algorithm B returns the share ciphertext $CT_{AU}^{w} = (A, B) = (PK_{AU}^{r}$, $H_2(e(PK_S, H_1(w))^r))$.

- Search token generation oracle O_{st}: With the input of a keyword w and a public key PK_{AU} by the adversary, B randomly chooses and calculates $T_1 = g^{r'}$ and $T_2 = H_1(w')^{1/x_{AU}} \cdot H(PK_S^{r'})$. B responds A_4 with $T_w = (T_1, T_2)$.

- Test oracle O_{te}: With the input of (CT_{AU}^{w} , $T_{w'}$) by the adversary, the algorithm B outputs 1 if the equation $B = H_2(e(A, \tau^a)$ holds or 0 otherwise.

Challenge: Once the Phase1 is finish, A_4 inputs two distinct keywords (w_0, w_1) where $|w_0| = |w_1|$ and a public key PK_{AU}^*. B randomly picks $c \in \{0, 1\}$ and sets $T_1^* = g^{b/l}$ and $T_2^* = H_1(w_c)^{1/x_{AU}^*} \cdot \eta$. If $\eta = H(g^{ab})$ then $T_{w_c}^*$ is a valid challenge token. If $r^* = b / l$, then $T_1^* = g^{r^*}$, $T_2^* = H_1(w_c)^{1/x_{AU}^*} \cdot \eta = H_1(w_c)^{1/SK_{AU}^*} \cdot H(g^{ab}) = H_1(w_c)^{1/SK_{AU}^*} \cdot H(g^{al \cdot (b/l)}) = H_1(w_c)^{1/SK_{AU}^*} H(PK_S^{r'})$. Finally, it responds with the challenge token $T_{w_c}^* = (T_1^*, T_2^*)$.

Phase 2: A_4 issues more queries, but with the restrictions as defined in Game 4.

Guess: At last, A_4 outputs a guess $c' \in \{0, 1\}$. If $c = c'$, B outputs 1, meaning that $\eta = H(g^{ab})$or 0 otherwise.

Similar to the proof of Theorem 3, it can be deduced that the lower bound on B's advantage is ε.

5. Performance Evaluation

We first compare the properties of the existing KU-PEKS framework [34] and the KU-CS-PEKS framework. The details are shown in **Table 1**. From the table, it is easy to see that the KU-CS-PEKS framework enjoys some good properties such as supporting privacy-preserving ciphertext update on the storage server and transmitting the search tokens without secure channel, while providing ciphertext sharing function.

Table 1. Properties of the KU-PEKS framework and the KU-CS-PEKS framework

Frameworks	Supporting key update	Supporting privacy-preserving ciphertext update	Supporting ciphertext sharing	Considering search token privacy	No secure channel
KU-PEKS	yes	no	no	no	no
KU-CS-PEKS	yes	yes	yes	yes	yes

Table 2. Computational efficiency of the compared schemes

Schemes	*KWCipherGen*	*CiphertextUpdate*	*CiphertextShare*	*SearchTokenGen*	*Test*
KU-PEKS	$4T_E + T_H + T_{MP} + T_P + T_M$	$5T_E + T_H + T_{MP} + T_P + T_M + T_I$	-	$T_{MP} + T_E$	$T_H + T_P$
KU-CS-PEKS	$T_E + T_{ET} + T_H + T_{MP} + T_P$	T_E	T_E	$T_H + T_{MP} + 3T_E$	$2T_H + T_P + 2T_E$

Table 3. Communicational efficiency of the compared schemes

Schemes	Length of a keyword ciphertext	Length of a search token
KU-PEKS	$3\|G\| + l$	$\|G\|$
KU-CS-PEKS	$\|G\| + l$	$2\|G\|$

Table 4. Time cost of each basic operation and bit-length of an element/hash value

Running time (ms)							Bit-length (bit)	
T_H	T_P	T_{MP}	T_E	T_{ET}	T_M	T_I	$\|G\|$	l
0.005	1.6	4.3	0.2	0.008	0	0.002	512	256

Table 2 and **Table 3** respectively give the computational efficiency comparison and the communicational efficiency comparison of the KU-PEKS scheme [34] and the proposed KU-CS-PEKS scheme, where T_H, T_P, T_{MP}, T_E, T_{ET}, T_M and T_I are the running time of a cryptographic hash operation, a bilinear pairing operation, a map-to-point encoding operation, an exponent operation in G, an exponent operation in G_T, a modular multiplication operation in Z_p and a modular inverse operation in Z_p respectively, l and $|G|$ respectively denote the bit-size of a hash value and the bit-size of an element in G.

To evaluate the computational efficiency, the schemes are tested by the PBC library [40]. The experiments are carried out on a laptop and Linux OS with an Intel Core i5-4210U CPU of 1.7GHz and 3.0GB RAM. The bilinear pairing is simulated by the Type-A pairing, which is defined on the curve $y^2 = x^3 + x$ over the finite field F_q for prime $q \equiv 3 \bmod 4$. Besides, the hash functions are implemented by SHA-256. **Table 4** gives the time cost of each basic operation and bit-length of a group element or a hash value.

The experimental results (see **Fig. 1** ~ **Fig. 4**) show that the costs of the *SearchTokenGen* algorithm and the *Test* algorithm in the KU-CS-PEKS scheme are higher than that of the KU-PEKS scheme, but the costs of the *KWCiphertextGen* algorithm and the *CiphertextUpdate* algorithm are lower than that of the KU-PEKS scheme. Specifically, the time for encrypting a keyword in the KU-CS-PEKS scheme is about 6.113*ms*, while that in the KU-PEKS scheme is about 6.705*ms*. The time for updating a keyword ciphertext in the KU-CS-PEKS scheme is about 0.2*ms*, while that in the KU-PEKS scheme is about 6.907*ms*. The time for generating a search token in the KU-CS-PEKS scheme is about 4.905*ms*, while that in the KU-PEKS scheme is about 4.5*ms*. In addition, the time for matching a keyword ciphertext with a search token in the KU-CS-PEKS scheme is about 2.01*ms*, while that in KU-PEKS scheme is about 1.605ms.

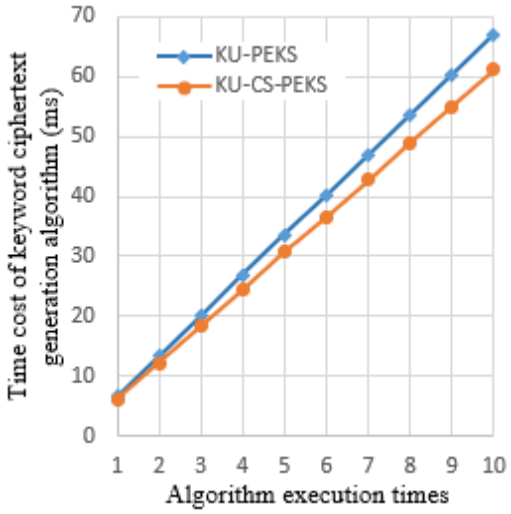


Fig. 1. Time cost of keyword ciphertext generation algorithm

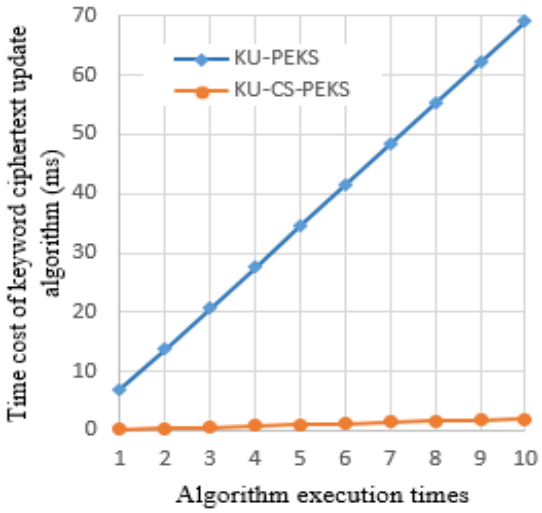


Fig. 2. Time cost of keyword ciphertext update algorithm

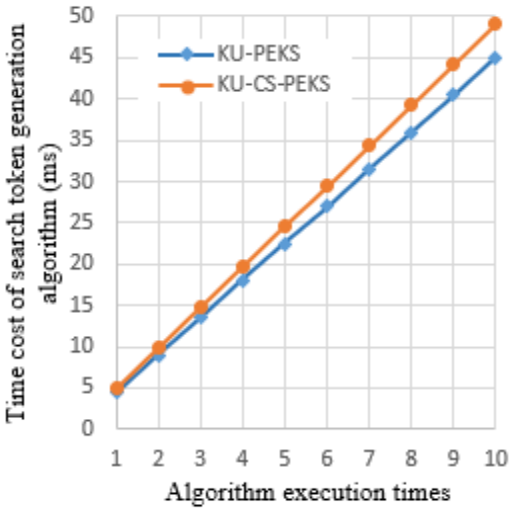


Fig. 3. Time cost of search token generation algorithm

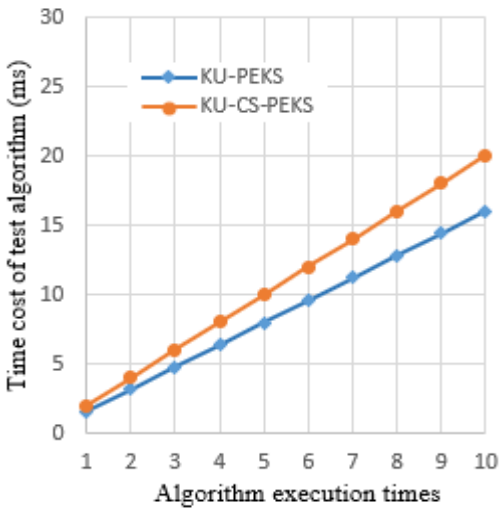


Fig. 4. Time cost of test algorithm

Regarding the communication overhead (as illustrated in **Fig. 5**), the keyword ciphertext length in KU-CS-PEKS scheme is 768 bits, while the keyword ciphertext length in the KU-PEKS scheme is 1792 bits. Besides, the size of a search token in KU-CS-PEKS scheme is 1024 bits, while that in the KU-PEKS scheme is 512 bits. Compared with the KU-PEKS scheme, the KU-CS-PEKS scheme has shorter keyword ciphertext and longer search token.

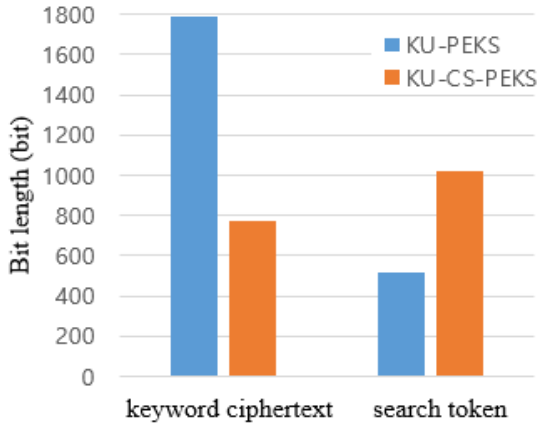


Fig. 5. Bit-length of keyword ciphertext and search token

To remove secure channels and achieve search token indistinguishability, the KU-CS-PEKS scheme has to generate longer search tokens and consumes more time in search token generation and match testing. However, the additional costs are worthwhile, because the proposed scheme effectively fixes the inherent security weaknesses in the previous KU-PEKS scheme.

6. Conclusion

This paper presents a PEKS framework, named key-updatable and ciphertext-sharable PEKS (KU-CS-PEKS). After formally defining its security (including the keyword ciphertext indistinguishability and the search token indistinguishability), a concrete KU-CS-PEKS scheme that is proven secure in the random oracle model is given. The KU-CS-PEKS scheme realizes privacy-preserving ciphertext update on the storage server and ciphertext sharing functions, while removing the secure channel requirement. The experimental results and comparisons demonstrate that the KU-CS-PEKS scheme is feasible and applicable.

References

[1] D. X. Song, D. Wagner, and A. Perrig, “Practical techniques for searches on encrypted data,” in *Proc. of IEEE Symp. Secur. Privacy*, pp. 44–55, May, 2000. Article (CrossRef Link)

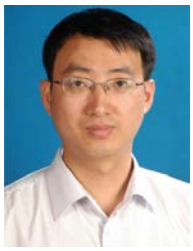
[2] P. Golle, J. Staddon, and B. Waters, “Secure conjunctive keyword search over encrypted data,” in *Proc. of ACNS*, pp. 31–45, 2004. Article (CrossRef Link)

[3] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, “Searchable symmetric encryption: Improved definitions and efficient constructions,” in *Proc. of 13th ACM Conf. Comput. Commun. Secur.*, pp. 79–88, Oct. 2006. Article (CrossRef Link)

[4] Z. Fu, X. Wu, C. Guan, X. Sun, and K. Ren, "Toward efficient multi-keyword fuzzy search over encrypted outsourced data with accuracy improvement," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 12, pp. 2706–2716, July. 2016. Article (CrossRef Link)
[5] Z. Xia, X. Wang, X. Sun, and Q. Wang, "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 2, pp. 340–352, Feb. 2015. Article (CrossRef Link)
[6] X. Liu, G. Yang, Y. Mu and R. H. Deng, "Multi-User Verifiable Searchable Symmetric Encryption for Cloud Storage," *IEEE Trans. Dependable Secure Comput.*, vol. 17, no. 6, pp. 1322-1332, 2020. Article (CrossRef Link)
[7] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. of Int. Conf. Theory Appl. Cryptograph. Techn*, pp. 506–522, 2004. Article (CrossRef Link)
[8] C. Gu, Y. Zhu, and H. Pan, "Efficient public key encryption with keyword search schemes from pairings," in *Proc. of Int. Conf. Inf. Secur. Cryptol*, pp. 372–383, 2008. Article (CrossRef Link)
[9] B. Zhang and F. Zhang, "An efficient public key encryption with conjunctive-subset keywords search," *J. Netw. Comput. Appl.*, vol. 34, no. 1, pp. 262–267, Jan. 2011. Article (CrossRef Link)
[10] D. J. Park, K. Kim, and P. J. Lee, "Public key encryption with conjunctive field keyword search," in *Proc. of Int. Workshop Inf. Secur. Appl*, pp. 73–86, 2004. Article (CrossRef Link)
[11] J. Baek, R. Safavi-Naini, and W. Susilo, "On the integration of public key data encryption and public key encryption with keyword search," in *Proc. of Int. Conf. Inf. Secur*, pp. 217–232, 2006. Article (CrossRef Link)
[12] Q. Tang and L. Chen, "Public-key encryption with registered keyword search," in *Proc. of Eur. Public Key Infrastruct*, pp. 163–178, Sep. 2009. Article (CrossRef Link)
[13] Q. Dong, Z. Guan, L. Wu, and Z. Chen, "Fuzzy keyword search over encrypted data in the public key setting," in *Proc. of Int. Conf. Web-Age Inf. Manage*, pp. 729–740, June. 2013. Article (CrossRef Link)
[14] J. Shi, J. Lai, Y. Li, R. H. Deng, and J. Weng, "Authorized keyword search on encrypted data," in *Proc. of Eur. Symp. Res. Comput. Secur*, pp. 419–435, 2014. Article (CrossRef Link)
[15] Z. Lv, C. Hong, M. Zhang, and D. Feng, "Expressive and secure searchable encryption in the public key setting," in *Proc. of Int. Conf. Inf. Secur*, pp. 364–376, 2014. Article (CrossRef Link)
[16] Y. Chen, J. Zhang, D. Lin, and Z. Zhang, "Generic constructions of integrated PKE and PEKS," *Des. Codes Cryptogr.*, vol. 78, no. 2, pp. 493–526, Feb. 2016. Article (CrossRef Link)
[17] H. Cui, Z. Wan, R. Deng, G. Wang, and Y. Li, "Efficient and expressive keyword search over encrypted data in cloud," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 3, pp. 409–422, May. 2018. Article (CrossRef Link)
[18] L. Li, C. Xu, X. Yu, B. Dou and C. Zuo, "Searchable encryption with access control on keywords in multi-user setting," *Journal of Cyber Security*, vol. 2, no. 1, pp. 9–23, Jan. 2020. Article (CrossRef Link)
[19] X. Li, F. Li, J. Jiang, and X. Mei, "Paillier-based fuzzy multi-keyword searchable encryption scheme with order-preserving," *Computers, Materials & Continua*, vol. 65, no. 2, pp. 1707–1721, Jan. 2020. Article (CrossRef Link)
[20] J. Baek, R. Safavi-Naini, and W. Susilo, "Public key encryption with keyword search revisited," in *Proc. of Int. Conf. Comput. Sci. Appl*, pp. 1249–1259, 2008. Article (CrossRef Link)
[21] H. S. Rhee, J. H. Park, W. Susilo, and D. H. Lee, "Improved searchable public key encryption with designated tester," in *Proc. of 4th Int. Symp. Inf., Comput., Commun. Secur*, pp. 376–379, Jan. 2009. Article (CrossRef Link)
[22] H. S. Rhee, J. H. Park, W. Susilo, and D. H. Lee, "Trapdoor security in a searchable public-key encryption scheme with a designated tester," *J. Syst. Softw.*, vol. 83, no. 5, pp. 763–771, May. 2010. Article (CrossRef Link)
[23] C. Hu and P. Liu, "A secure searchable public key encryption scheme with a designated tester against keyword guessing attacks and its extension," in *Proc. of Int. Conf. Comput. Sci., Environ., Ecoinform., Educ*, vol. 215, pp. 131-136, 2011. Article (CrossRef Link)

[24] H. S. Rhee, J. H. Park, and D. H. Lee, “Generic construction of designated tester public-key encryption with keyword search,” *Inf. Sci*., vol. 205, no. 1, pp. 93–109, Nov. 2012. Article (CrossRef Link)
[25] L. Fang, W. Susilo, C. Ge, and J. Wang, “Public key encryption with keyword search secure against keyword guessing attacks without random oracle,” *Inf. Sci*., vol. 238, pp. 221–241, July. 2013. Article (CrossRef Link)
[26] Y. Lu, G. Wang, and J. Li, “Keyword guessing attacks on a public key encryption with keyword search scheme without random oracle and its improvement,” *Inf. Sci*., vol. 479, pp. 270–276, Apr. 2019. Article (CrossRef Link)
[27] T. Wu, T. Tsai, and Y. Tseng, “Efficient searchable id-based encryption with a designated server,” *Ann. Telecommun.*, vol. 69, no. 7–8, pp. 391–402, Aug. 2014. Article (CrossRef Link)
[28] Y. Lu, G. Wang, J. Li, and J. Shen, “Efficient designated server identity-based encryption with conjunctive keyword search,” *Ann. Telecommun.*, vol. 72, no. 5–6, pp. 359–370, June 2017. Article (CrossRef Link)
[29] J. Liu, J. Lai, and X. Huang, “Dual trapdoor identity-based encryption with keyword search,” *J. Soft Comput.*, vol. 21, no. 10, pp. 2599-2607, May 2017. Article (CrossRef Link)
[30] Y. Peng, J. Cui, C. Peng, and Z. Ying, “Certificateless public key encryption with keyword search,” *China Commun.*, vol. 11, no. 11, pp. 100–113, Nov. 2014. Article (CrossRef Link)
[31] Y. Lu, J. Li, and F. Wang, “Pairing-free certificate-based searchable encryption supporting privacy-preserving keyword search function for IIoTs,” *IEEE. Trans. Industr. Inform*., vol. 17, no. 4, pp. 2696–2706, Apr. 2021. Article (CrossRef Link)
[32] X. Yu, C. Xu, L. Xu and Y. Wang, “Lattice-based searchable encryption scheme against inside keywords guessing attack,” *Computers, Materials & Continua*, vol. 64, no. 2, pp. 1107–1125, 2020. Article (CrossRef Link)
[33] M. Ali, C. Xu and A. Hussain, "Authorized attribute-based encryption multi-keywords search with policy updating," *Journal of New Media*, vol. 2, no.1, pp. 31–43, Aug. 2020. Article (CrossRef Link)
[34] H. Anada, A. Kanaoka, N. Matsuzaki, Y. Watanabe, “Key-updatable public-key encryption with keyword search: models and generic constructions,” *Information Security and Privacy*, pp. 341–359, June. 2018. Article (CrossRef Link)
[35] J. Shao, Z. Cao, X. Liang and H. Lin, “Proxy re-encryption with keyword search,” *Inf. Sci*., vol. 180, no. 13, pp. 2576–2587, July. 2010. Article (CrossRef Link)
[36] R. Canetti, S. Hohenberger, “Chosen-ciphertext secure proxy re-encryption,” in *Proc. of ACM CCS*, pp. 185–194, Oct. 2007. Article (CrossRef Link)
[37] Y. Liu, Y. Ren, Q. Wang and J. Xia, "The development of proxy re-encryption," *Journal of Cyber Security*, vol. 2, no. 1, pp. 1–8, Jan. 2020. Article (CrossRef Link)
[38] D. Boneh, X. Boyen, “Efficient selective-ID secure identity-based encryption without random oracle,” in *Proc. of Int. Conf. Theory Appl. Cryptograph. Techn*, pp. 223–238, 2004. Article (CrossRef Link)
[39] M. Abdalla, M. Bellare, P. Rogaway, “The oracle Diffie–Hellman assumptions and an analysis of DHIES,” in *Proc. of Naccache, D.* (*ed*.) *CT-RSA* 2001, vol. 2020, pp. 143–158. April. 2001. Article (CrossRef Link)
[40] B. Lynn, PBC library: The Pairing-Based Cryptography Library, 2013. [Online]. Available: http://crypto.stanford.edu/pbc.

Fen Wang received the B.S. degree in computer science from Maanshan University, Anhui, China, in 2018. She is working toward the M.S. degree in computer science in Nanjing Normal University, Nanjing, China. Her research interests include cryptography and information security.



Yang Lu received the Ph.D. degree in computer science from PLA University of Science and Technology, Nanjing, China, in 2009. He is currently a Professor with the School of Computer Science and Technology, Nanjing Normal University, Nanjing, China. His research interests include cryptography and information security, cloud computing, etc.

Zhongqi Wang received the B.S. degree in information management and information system from Nanjing Normal University, Nanjing, China, in 2020. Now, he is a research student of the Risk and Resilience Program of Graduate School of Science and Technology, University of Tsukuba, Japan. His research interests include information security, cryptography theory and application.

Jinmei Tian received the B.S. degree in computer science from Hainan Normal University, Hainan, China, in 2019. She is working toward the M.S. degree in computer science in Nanjing Normal University, Nanjing, China. Her research interests include cryptography and information security.