Proxy Re-encryption with Keyword Search: New Definitions and Algorithms with Proofs*

Wei-Chuen Yau[1], Raphael C.-W. Phan[2,†], Swee-Huay Heng[3], and Bok-Min Goi[4]
[1]*Faculty of Engineering, Multimedia University, Malaysia.*
wcyau@mmu.edu.my
[2]*Faculty of Engineering, Loughborough University, England.*
r.phan@lboro.ac.uk
[3]*Faculty of Information Science & Technology, Multimedia University, Malaysia.*
shheng@mmu.edu.my
[4]*Faculty of Engineering & Science, Tunku Abdul Rahman University, Malaysia.*
goibm@utar.edu.my

Abstract

We propose a new definition for searchable proxy re-encryption scheme (Re-PEKS), define the first known searchable proxy re-encryption scheme with a designated tester (Re-dPEKS), and then give concrete constructions of both Re-PEKS and Re-dPEKS schemes that are secure in the random oracle model, along with the proofs.

Keywords: *Searchable proxy re-encryption, public key encryption with keyword search, random oracle model.*

1. Introduction

Public key encryption with keyword search (PEKS) schemes enable searching of keywords within encrypted messages. These schemes are desirable for mobile devices such as smartphones for selectively downloading encrypted messages from gateways, e.g. accessing to emails while on mobile internet. Consider an e-mail system that consists of three entities, namely, a sender (Bob), a receiver (Alice), and a server (email gateway). Bob sends an encrypted message M appended with some encrypted keywords $w_1,..., w_n$ that are associated with the message to the email gateway in the following format:

$$PKE(pk_A, M) \| PEKS(pk_A, w_1) \| ... \| PEKS(pk_A, w_n),$$

where PKE is a standard public key encryption scheme and pk_A is the public key of Alice. Alice can give the email gateway a trapdoor associated with a searching keyword w. The PEKS scheme enables the gateway to test whether w is a keyword associated with the email but learns nothing else about the email.

The first PEKS scheme was proposed by Boneh et al. in 2004 [6]. Baek et al. [4] proposed a Secure Channel Free Public Key Encryption with Keyword Search (SCF-PEKS) scheme to remove the requirement of a secure channel for sending a trapdoor in PEKS of [6]. This scheme is also known as a PEKS scheme with a designated tester (dPEKS). A dPEKS

* Research supported by BGM Fund (MOSTI/BGM/R&D/500-2/8). This is the full version of our SecTech'10 paper [21], with proofs for our constructions.
† Part of this work done while the author was visiting Multimedia University.

scheme ensures that no one except the designated server is able to run the test function (dTest). Some other PEKS and dPEKS schemes in the literature include [1, 3, 4, 10-12, 14-17, 22].

In some situations, Alice may need to delegate her decryption right to her assistant Carol. For such scenario, the email gateway needs to convert encrypted emails for Alice into ciphertexts which can be decrypted by Carol. This function can be achieved with proxy re-encryption (PRE) which was first introduced in [5]. More precisely, a proxy re-encryption scheme enables a semi-trusted proxy to convert a ciphertext encrypted under Alice's public key into a ciphertext of the same message for Carol with a given special information (i.e., a re-encryption key). However, the proxy should not learn secret keys of Alice or Carol and the plaintext during the conversion.

Since the original keyword ciphertext encrypted under Alice's public key cannot be tested with trapdoor generated by Carol, it is natural to ask how to enable searching of re-encrypted emails in the above-mentioned scenario. We need a way that enables Carol to instruct the gateway to search for those re-encrypted messages associated to certain keywords with her secret key. To do so, the gateway performs the re-encryption of message ciphertext with a standard proxy re-encryption (PRE) scheme, and re-encryption of keyword ciphertexts using a proxy re-encryption with keyword search (Re-PEKS) or searchable proxy re-encryption scheme. i.e.,

$$PRE(rk, C) \,||\, Re-PEKS(rk, C_{w_1}) \,||\, ... \,||\, Re-PEKS(rk, C_{w_n})$$

where rk is a re-encryption key to re-encrypt Alice's ciphertext into ciphertext for Carol, C is a message ciphertext encrypted with PKE, and $C_{w_1},...,C_{w_n}$ are keyword ciphertexts encrypted with PEKS. Shao et al. proposed a proxy re-encryption with keyword search (they called it as PRES) scheme [19] based on [7, 9]. However, the formation of their scheme is different from the above formation of Re-PEKS scheme.

1.1. Our Contributions

In this paper, we propose a new definition for searchable proxy re-encryption scheme (Re-PEKS). Compared with the definition of proxy re-encryption with keyword search scheme (PRES) in [19], our definition has the following differences: Shao et al.'s PRES scheme encrypts the message and keyword in the same encryption algorithm. Our definition extends the original PEKS definition [6] by including the algorithms of re-encryption key generation and re-encryption of keyword ciphertext. This approach keeps the encryption of message and encryption of keyword separate so that we can have the flexibility to select which standard PRE and Re-PEKS schemes to be used for satisfying the requirements of the actual applications . For example, we may combine a more efficient encryption function (Re-PEKS) used for keyword while encryption of the message uses standard PRE techniques.

A more substantial contribution of our work is that we define the first searchable proxy re-encryption scheme with a designated tester (Re-dPEKS). This scheme allows a proxy (e.g., an email gateway) with a re-encryption key to translate an keyword w encrypted under public key pk_A into the same keyword encrypted under a different public key pk_B. In addition, only a designated email gateway can test whether or not a given dPEKS cipheretext (encrypted keyword) is associated with an email upon receiving a trapdoor by using its private key, but learns nothing else about the email.

We give concrete constructions of both Re-PEKS and Re-dPEKS schemes and prove their security in the random oracle model under bilinear Diffie-Hellman (BDH) assumption.

1.2. Paper Organization

The rest of the paper is organized as follows. In section 2, we briefly review the properties of re-encryption schemes and hardness assumptions used in our proofs. We give definitinions, security models, constructions, and security proofs of our Re-PEKS and Re-dPEKS schemes in section 3 and 4 respectively. Finally, section 5 lists open research problems and provides our conclusions.

2. Preliminaries

2.1. Properties of Re-encryption Schemes

We briefly describe some of the properties for re-encryption schemes [2] that are related to our proposed schemes:

- **Unidirectional:** Delegation from user X to user Y only allows re-encryption in one direction, i.e., $X \rightarrow Y$. If it allows re-encryption in both directions, i.e., from X to Y and vice versa ($X \leftrightarrow Y$), then the scheme is bidirectional. Our proposed scheme is bidirectional.
- **Multi-use:** A re-encryption ciphertext from user X to user Y can be re-encrypted again from user Y to user Z and so on, i.e., it can be re-encrypted multiple times. Our proposed scheme is multi-use.
- **Collusion safe:** Collusion of user X and proxy can recover user Y's secret key. Our scheme is not collusion safe, i.e., we assume no collusion between proxy and users.

2.2. Bilinear Maps and Complexity Assumptions

We briefly describe mathematical background and complexity assumptions that used throughout this paper.

- **Bilinear maps:** Let G and G_T denote two cyclic groups of prime order q. A bilinear map $e: \mathsf{G} \times \mathsf{G} \rightarrow \mathsf{G}_T$ satisfies the following properties:
  - Bilinearity: For all $g \in \mathsf{G}$ and $a, b \in \mathsf{Z}_q$, $e(g^a, g^b) = e(g, g)^{ab}$.
  - Non-degeneracy: There exists a $g \in \mathsf{G}$ such that $e(g, g) \neq 1$.
  - Computability: There is an efficient algorithm to compute the map e.
- **Bilinear Diffie-Hellman (BDH) problem:** The BDH problem [7] is as follows: given $g, g^a, g^b, g^c \in \mathsf{G}$ as input, compute $e(g, g)^{abc} \in \mathsf{G}_T$. We say that BDH assumption holds if all polynomial time algorithms have a negligible advantage in solving the BDH problem.
- **Modified Bilinear Diffie-Hellman (mBDH) problem:** The mBDH problem [18] is as follows: given $g, g^a, g^b, g^c \in \mathsf{G}$ as input, compute $e(g, g)^{ab/c} \in \mathsf{G}_T$. We say that mBDH assumption holds if all polynomial time algorithms have a negligible advantage in solving the mBDH problem. The mBDH problem is equivalent to the BDH problem, see the proof in [9] for the decisional variant of the assumption.

3. Bidirectional Re-PEKS Scheme

3.1. Definition

We define searchable proxy re-encryption scheme (Re-PEKS) and only consider bidirectional, multi-use Re-PEKS.

Definition 1. ***(Bidirectional, Multi-use Re-PEKS)*** *A bidirectional, multi-use, proxy re-encryption with keyword search (Re-PEKS) scheme consists of the following algorithms:*

- Setup(1^k): On input a security parameter 1^k, it returns a public parameter, PP.
- KeyGen(PP): On input PP, it returns a public-private key pair $[pk, sk]$.
- ReKeyGen(PP, sk_i, sk_j): On input PP, a private key sk_i, and a private key sk_j, where $i \neq j$, it returns a re-encryption key $rk_{i \leftrightarrow j}$.*
- PEKS(PP, pk_i, w): On input PP, pk_i, and a keyword $w \in$ keyword space KW, it returns a PEKS ciphertext $C_{i,w}$ of w.
- RePEKS(PP, $rk_{i \leftrightarrow j}$, $C_{i,w}$): On input PP, $rk_{i \leftrightarrow j}$, and an original PEKS ciphertext $C_{i,w}$, it returns a re-encryption PEKS ciphertext $C_{j,w}$ of w for receiver j.
- Trapdoor(PP, sk_i, w): On input PP, sk_i, and a keyword w, it returns a trapdoor $T_{i,w}$.
- Test(PP, C, T_w): On input PP, a PEKS ciphertext C = PEKS(PP, pk, w'), and a trapdoor T_w = Trapdoor(PP, sk, w), it returns 1 if $w = w'$ and 0 otherwise.

Correctness: Let key pairs $[pk_i, sk_i]$ and $[pk_j, sk_j] \leftarrow \mathsf{KeyGen}(\mathsf{PP})$,

$rk_{i \leftrightarrow j} \leftarrow \mathsf{ReKeyGen}(\mathsf{PP}, sk_i, sk_j)$, $C_{i,w'} \leftarrow \mathsf{PEKS}(\mathsf{PP}, pk_i, w')$,

$T_{i,w} \leftarrow \mathsf{Trapdoor}(\mathsf{PP}, sk_i, w)$, $T_{j,w} \leftarrow \mathsf{Trapdoor}(\mathsf{PP}, sk_j, w)$, $\forall w, w' \in$ keyword space KW, it holds that

- Test($\mathsf{PP}, C_{i,w'}, T_{i,w}$) = 1 if $w = w'$, and 0 otherwise.
- Test($\mathsf{PP}, \mathsf{PEKS}(\mathsf{PP}, rk_{i \leftrightarrow j}, C_{i,w'}), T_{j,w}$) = 1 if $w = w'$, and 0 otherwise.

We note that our searchable proxy re-encryption scheme (Re-PEKS) is an extended searchable public key encryption scheme (PEKS). In particular, we can add two algorithms, i.e., ReKeyGen and RePEKS, in a PEKS scheme to form a Re-PEKS scheme.

3.2. Security Model

The security of a Re-PEKS scheme requires that the adversary should not be able to distinguish which keyword corresponds to a given ciphertext without the trapdoor from the target receiver or a delegatee.

Bidirectional, Multi-use Re-PEKS CKA-Security Game. We use the following game between a challenger and an active adversary A to define the security for the Re-PEKS scheme. The game consists of the following phases, which are executed in order. The oracles

* This algorithm becomes noninteractive if the input of sk_j is replaced by public key pk_j, where the delegatee does not involve in the generation of re-encryption key.

in each phase can be executed poly(k) times in any order unless otherwise specified. We assume a static corruption model: i.e., adversary has to determine either corrupt a party or not at the time the key pair of each party is generated.

1. Game Setup:
   - **Public Parameter Generation**: The challenger runs Setup(1^k) to generate the public parameter PP and gives it to the adversary A. This oracle is executed first and only once.
   - **Uncorrupted Receiver Key Generation**: The challenger runs $\mathsf{KeyGen}(\mathsf{PP})$ and returns a public-private key pair $[pk, sk]$. It gives pk to A. Let L_H be the set of honest receiver indices.
   - **Corrupted Receiver Key Generation**: The challenger runs $\mathsf{KeyGen}(\mathsf{PP})$ and returns a public-private key pair $[pk, sk]$. It gives $[pk, sk]$ to A. Let L_C be the set of corrupt receiver indices.
2. Phase 1: A makes the following queries:
   - **Trapdoor Generation** O_{td}: On input (i, w) by the adversary, where $i \in \mathsf{L}_H \cup \mathsf{L}_C$, $w \in$ keyword space KW, the challenger runs $\mathsf{Trapdoor}(\mathsf{PP}, sk_i, w)$ and returns a trapdoor $T_{i,w}$ associated with keyword w which generated by secret key of user i to A.
   - **Re-encryption Key Generation** O_{rk}: On input (i, j) by the adversary, where $i \neq j$, the challenger runs $\mathsf{ReKeyGen}(\mathsf{PP}, sk_i, sk_j)$ and returns a re-encryption key $rk_{i \leftrightarrow j}$ to A. We restrict that either both i and j are corrupted or both are uncorrupted, i.e., $i, j \in \mathsf{L}_H$ or $i, j \in \mathsf{L}_C$. In another words, re-encryption key queries between a corrupted and an uncorrupted party are not allowed.
   - **Re-encryption** O_{renc}: On input (i, j) and an original PEKS ciphertext $C_{i,w}$ by the adversary, both either from L_H or L_C, the challenger returns the re-encrypted PEKS ciphertext $C_{j,w}$ = $\mathsf{RePEKS}(\mathsf{PP}, \mathsf{ReKeyGen}(\mathsf{PP}, sk_i, sk_j), C_{i,w})$. All re-encryption queries where $i = j$ or where $i \in \mathsf{L}_H$ and $j \in \mathsf{L}_C$ or where $i \in \mathsf{L}_C$ and $j \in \mathsf{L}_H$ are ignored, i.e., an output of $\perp$.
3. Challenge: On input (i^*, w_0, w_1) by the adversary, where $w_0, w_1 \in$ keyword space KW, the challenger picks a random $b \in \{0,1\}$ and returns the challenge ciphertext $C^* = \mathsf{PEKS}(\mathsf{PP}, pk_{i^*}, w_b)$ to A. The restriction is that $i^* \in \mathsf{L}_H$ and A did not previously ask for the trapdoors of $(i^*, w_0), (i^*, w_1)$ from O_{td}. In addition, A did not previously ask for the re-encryption key of (i^*, j) or (j, i^*) from O_{rk} and the trapdoors of $(j, w_0), (j, w_1)$ from O_{td}.
4. Phase 2: *The adversary is allowed to ask the same types of queries as in Phase 1, except the following queries:*
   - $\mathsf{O}_{td}(i^*, w_b)$, where $b \in \{0,1\}$.
   - $\mathsf{O}_{rk}(i^*, j)$ and $\mathsf{O}_{td}(j, w_b)$, or $\mathsf{O}_{rk}(j, i^*)$ and $\mathsf{O}_{td}(j, w_b)$, where $b \in \{0,1\}$.
   - $\mathsf{O}_{renc}(i^*, j, C^*)$ and $\mathsf{O}_{td}(j, w_b)$, where $b \in \{0,1\}$.

5. Guess: Finally, A outputs a guess $b' \in \{0,1\}$ and wins the game if $b = b'$.

We define A's advantage in breaking the Re-PEKS scheme as:

$$Adv_{\mathsf{A}}^{\text{IND-CKA Re-PEKS}}(k) = |\Pr[b = b'] - 1/2|.$$

Definition 2. *We say that a Re-PEKS scheme is indistinguishability against an adaptive chosen keyword attack (IND-CKA) if for any polynomial time attacker* A *we have that* $Adv_{\mathsf{A}}^{\text{IND-CKA Re-PEKS}}(k)$ *is a negligible function.*

3.3. Construction

We construct a secure bidirectional, multi-use Re-PEKS scheme based on PEKS scheme in [6], except a slight modification in the structure of PEKS ciphertext and trapdoor. In addition, we use the re-encryption technique in [9, 5]. These techniques are similar to the construction of PRES scheme in [19].

- Setup(1^k): Let G and G_T be bilinear groups of order q. Let $H_1 : \{0,1\}^* \to \mathsf{G}$, and $H_2 : \mathsf{G}_T \to \{0,1\}^k$ be independent hash functions. Given a security parameter k, the algorithm picks a random generator $g \in \mathsf{G}$. It returns a public parameter $\mathsf{PP} = (q, \mathsf{G}, \mathsf{G}_T, g, e, H_1, H_2)$.
- KeyGen(PP): On input PP, select a random value $x \in \mathsf{Z}_q$. Set receiver's public key $pk = g^x$ and private key $sk = x$. Return $[pk, sk]$.
- ReKeyGen(PP, sk_X, sk_Y): On input $sk_X = x$ and $sk_Y = y$, output the bidirectional re-encryption key $rk_{X \leftrightarrow Y} = y/x \mod q$.*
- PEKS(PP, pk_X, w): To encrypt a keyword $w \in$ keyword space KW under receiver's public key $pk_X = g^x$, select a random value $r \in \mathsf{Z}_q$ and compute the PEKS ciphertext $C_{X,w} = [A, B] = [g^{xr}, H_2(e(g^r, H_1(w)))]$.
- RePEKS(PP, $rk_{X \leftrightarrow Y}$, $C_{X,w}$): On input a re-encryption key $rk_{X \leftrightarrow Y}$, and a PEKS ciphertext $C_{X,w} = [A, B]$, compute $A' = A^{rk_{X \leftrightarrow Y}} = g^{(xr)(y/x)} = g^{yr}$. Output the re-encrypted PEKS ciphertext from receiver X to Y as $C_{Y,w} = [A', B]$.
- Trapdoor(PP, sk_X, w): On input a receiver's private key $sk_X = x$ and a keyword w, output the trapdoor as $T_{X,w} = H_1(w)^{1/x}$.
- Test(PP, C, T_w): On input a PEKS ciphertext $C = [A, B]$ and a trapdoor T_w, check if $B = H_2(e(A, T_w))$. It outputs 1 if the above equality holds, and 0 otherwise.

* User X with private key x can delegate to user Y with private key y by selecting a random $r \in \mathsf{Z}_q$ and sending $rx \mod q$ to Y as well as r to proxy. Y sends $y/rx \mod q$ to proxy. The proxy computes re-encryption key $rk_{X \leftrightarrow Y} = (r)(y/rx) \mod q = y/x \mod q$. We assume that communications among proxy and users are via a secure channel. In addition, the scheme makes no security guarantee if the proxy colludes with either party [9].

Correctness:

Assume the PEKS ciphertext of keyword w' is $[A,B]=[g^{xr}, H_2(e(g^r, H_1(w')))]$ and the trapdoor associated to keyword w is $T_{X,w}=H_1(w)^{\frac{1}{x}}$. If $w=w'$, $B=H_2(e(g^r,H_1(w')))=H_2(e(g^{xr},H_1(w')^{\frac{1}{x}})=H_2(e(A,T_{X,w}))$. It is easy to verify that the correctness of the equation holds for multi-test as re-encrypted ciphertext has the same form as the original ciphertext.

3.4. Security Analysis

Theorem 1. *The Re-PEKS scheme is IND-CKA secure in the random oracle model assuming that the mBDH problem is intractable.*

Proof. Let A be a polynomial-time attack algorithm that has advantage ε in breaking the Re-PEKS scheme. Suppose A makes at most $q_{H_2}>0$ queries to the random oracle H_2 and at most $q_{td}>0$ trapdoor queries. We construct an algorithm B that has an advantage of $\varepsilon/(eq_{td}q_{H_2})$ in solving the mBDH problem in G, G_T, where e is a base of the natural logarithm. On mBDH input $(g, u_1=g^{\alpha}, u_2=g^{\beta}, u_3=g^{\gamma}\in\mathsf{G})$, B's goal is to compute $e(g,g)^{\alpha\beta/\gamma}$. B simulates the challenger and interacts with A as follows:

1. Game Setup:
   - **Public Parameter Generation**: B setups the public parameter PP and gives $\mathsf{PP}=(q,\mathsf{G},\mathsf{G}_T,g,e,H_1,H_2)$ to A.
   - **Uncorrupted Receiver Key Generation**: On input an index i, B selects a random $x_i\in\mathsf{Z}_q$, and outputs the public key $pk_i=u_3^{x_i}=(g^{\gamma})^{x_i}$, where the private key is implicitly defined as $sk_i=\gamma x_i$. It adds the tuple $\langle i, pk_i, x_i\rangle$ in L_H.
   - **Corrupted Receiver Key Generation**: On input an index i, B selects a random $x_i\in\mathsf{Z}_q$, and outputs $pk_i=g^{x_i}$ and $sk_i=x_i$. It adds the tuple $\langle i, pk_i, x_i\rangle$ in L_C.
2. Hash Function Queries: A can query the random oracle H_1 or H_2 any time.
   - H_1**-query** O_{H_1} : B maintains an H_1-list with tuples $\langle w_n, h_n, d_n, c_n\rangle$ which is initially empty. On input w_i, B responds as follows:
     - If the query w_i is found in the H_1-list with an entry $\langle w_i, h_i, d_i, c_i\rangle$, output $H_1(w_i)=h_i$.
     - Otherwise, B selects a random $d_i\in\mathsf{Z}_q$ and generates a random coin $c_i\in\{0,1\}$ so that Pr[$c_i=0$]=$1/(q_{td}+1)$.
       - If $c_i=1$, B computes $h_i=u_3^{d_i}=(g^{\gamma})^{d_i}$.
       - If $c_i=0$, B computes $h_i=u_1^{d_i}=(g^{\alpha})^{d_i}$

B adds the tuple $\langle w_i, h_i, d_i, c_i \rangle$ to H_1-list and returns $H_1(w_i) = h_i$.

- H_2 **-query** O_{H_2} : Similarly, B maintains an H_2 -list with tuples $\langle t, M \rangle$ which is initially empty. On input $t \in \mathsf{G}_\mathsf{T}$, B responds with $H_2(t) = M$. For each new t, B responds to the $H_2(t)$ query by selecting a new random value $M \in \{0,1\}^k$ and setting $H_2(t) = M$. B then adds the tuple $\langle t, M \rangle$ to the H_2 list.

3. Phase 1: When A issues the following queries, B responds as follows:

- **Trapdoor Generation**: On input (i, w_i) to O_{td}, do:
  - B gets the response from O_{H_1} to obtain an entry $\langle w_i, h_i, d_i, c_i \rangle$ in H_1-list. If $c_i = 0$, output $\perp$ and abort. Otherwise, we know $c_i = 1$ and we have $h_i = u_3^{d_i} = (g^\gamma)^{d_i} \in \mathsf{G}$ in the H_1-list.
  - If $i \in \mathsf{L}_C$, B sets $T_w = h_i^{1/x_i} = (g^\gamma)^{d_i/x_i}$, where x_i is obtained from $\langle i, pk_i, x_i \rangle$ in L_C
  - If $i \in \mathsf{L}_H$, B sets $T_w = g^{d_i/x_i}$, where x_i is obtained from the tuple $\langle i, pk_i, x_i \rangle$ in L_H. Observe that $H_1(w_i)^{1/\gamma x_i} = (g^{\gamma \cdot d_i})^{1/\gamma x_i} = g^{d_i/x_i}$ and therefore T_w is the correct trapdoor component for the keyword w_i under the implicitly defined user's private key γx_i.
  - B gives T_w to A.
- **Re-encryption Key Generation**: On input (i, j) to O_{rk}, do:
  - If (1) $i \in \mathsf{L}_H$ and $j \in \mathsf{L}_C$ or (2) $i \in \mathsf{L}_C$ and $j \in \mathsf{L}_H$ or (3) $i = j$ or (4) i or j not in L_H or L_C, output $\perp$.
  - Otherwise, output $rk_{i \leftrightarrow j} = x_j / x_i$.
- **Re-encryption**: On input $(i, j, C_{i,w})$ to O_{renc} where $C_{i,w}$ is an original PEKS ciphertext, do:
  - If (1) $i \in \mathsf{L}_H$ and $j \in \mathsf{L}_C$ or (2) $i \in \mathsf{L}_C$ and $j \in \mathsf{L}_H$ or (3) $i = j$ or (4) i or j not in L_H or L_C, output $\perp$.
  - If i and j are both from L_H or they are both from L_C, B obtains the re-encryption key $rk_{i \leftrightarrow j} = x_j / x_i$ from O_{rk} and returns the re-encryption ciphertext $C_{j,w} = \mathsf{RePEKS}(\mathsf{PP}, x_j/x_i, C_{i,w})$ to A.

4. Challenge: At some points, A gives the challenge tuple (i^*, w_0, w_1) to B, where $w_0, w_1 \in$ keyword space KW, do:

  - If (1) i^* is not from L_H or (2)the trapdoors of $(i^*, w_0), (i^*, w_1)$ from O_{td} was asked by A in Phase 1, or (3) the re-encryption key of (i^*, j) from O_{rk} and the trapdoors of $(j, w_0), (j, w_1)$ from O_{td} were asked by A in Phase 1, B returns $\perp$.

- Otherwise, B asks O_{H_1} to obtain h_0, $h_1 \in \mathsf{G}$ such that $H_1(w_0) = h_0$ and $H_1(w_1) = h_1$. Let $\langle w_b, h_b, d_b, c_b \rangle$, where $b \in \{0,1\}$, be the corresponding tuples in H_1-list. If both $c_0 = 1$ and $c_1 = 1$, B returns $\perp$ and aborts.
- Otherwise, at least one of c_0 or c_1 is equal to 0. B picks a random $b \in \{0,1\}$ such that $c_b = 0$.
- B selects a random $M^* \in \{0,1\}^k$ and sets $A^* = u_2^{x_{i^*}} = g^{\beta \cdot x_{i^*}}$ and $B^* = M^*$. Note that this challenge implicitly defines $A^* = (pk_{i^*})^{\beta/\gamma} = (g^{\gamma \cdot x_{i^*}})^{\beta/\gamma}$. Also, it defines $M^* = H_2(e(g^{\beta/\gamma}, H_1(w_b))) = H_2(e(g^{\beta/\gamma}, g^{\alpha d_b})) = H_2(e(g,g)^{(\alpha\beta/\gamma)(d_b)})$. Thus, $C^* = [A^*, B^*]$ is a valid PEKS ciphertext for w_b as required. B returns the challenge PEKS ciphertext C^* to A.

5. Phase 2: A is allowed to ask the same query types as in Phase 1 and B responds identically in Phase 1, except the following queries where B returns $\perp$:
   - $\mathsf{O}_{td}(i^*, w_b)$, where $b \in \{0,1\}$.
   - $\mathsf{O}_{rk}(i^*, j)$ and $\mathsf{O}_{td}(j, w_b)$, or $\mathsf{O}_{rk}(j, i^*)$ and $\mathsf{O}_{td}(j, w_b)$, where $b \in \{0,1\}$.
   - $\mathsf{O}_{renc}(i^*, j, C^*)$ and $\mathsf{O}_{td}(j, w_b)$, where $b \in \{0,1\}$.

6. Guess: Finally, A outputs a guess $b' \in \{0,1\}$. B picks a random pair $\langle t, M \rangle$ from the H_2-list and outputs t^{1/d_b} as its guess for $e(g,g)^{\alpha\beta/\gamma}$, where d_b is the value used in the Challenge step. Since A must have asked a query of either $H_2(e(g^{\beta/\gamma}, H_1(w_0)))$ or $H_2(e(g^{\beta/\gamma}, H_1(w_1)))$, the H_2-list contains a tuple where $t = e(g^{\beta/\gamma}, H_1(w_b)) = e(g,g)^{(\alpha\beta/\gamma)(d_b)}$ with probability 1/2. If B picks this tuple $\langle t, M \rangle$, then $t^{1/d_b} = e(g,g)^{\alpha\beta/\gamma}$ as required.

This completes the description of algorithm B. We now use the similar approach as in [6] to analyze the probability that B does not abort during the simulation. We define the following three events:

E_1: B does not abort as a result of any A's trapdoor queries.

E_2: B does not abort during the challenge phase.

E_3: A does not issue a query for either $H_2(e(g^r, H_1(w_0)))$ or $H_2(e(g^r, H_1(w_1)))$.

Claim 1: $\Pr[\mathsf{E}_1] \geq 1/e$

Proof. Without loss of generality we assume that A does not ask for the trapdoor of the same keyword twice. Prior to issuing the query, the bit c_i in the tuple $\langle w_i, h_i, d_i, c_i \rangle$ of H_1-list is independent of A's view. Since the only value that could be given to A that depends on c_i is $H(w_i)$ and the distribution of $H(w_i)$ is the same whether $c_i = 0$ or $c_i = 1$. Hence, $\Pr[\neg\mathsf{E}_1] \geq 1/(q_{td} + 1)$. Since A makes at most q_{td} queries, we have $\Pr[\mathsf{E}_1] \geq (1 - 1/(q_{td} + 1))^{q_{td}} \geq 1/e$.

Claim 2: $\Pr[\mathsf{E}_2] \geq 1/q_{td}$

Proof. B will abort during the challenge phase if $c_0 = c_1 = 1$. Since A has not queried $\mathsf{O}_{td}(i^*, w_i)$, where $i = 0,1$, both c_0 and c_1 are independent of A's current view. In addition, these two values are independent of one another. Therefore, we have $\Pr[\mathsf{E}_2] = 1 - \Pr[c_0 = c_1 = 1] = 1 - (1 - 1/(q_{td} + 1))^2 \geq 1/q_{td}$.

Since A can never issue a trapdoor query of the challenge keywords w_0, w_1, the two events E_1 and E_2 are independent. Hence, we have $\Pr[\mathsf{E}_1 \wedge \mathsf{E}_2] \geq 1/eq_{td}$.

Claim 3: $\Pr[\neg\mathsf{E}_3] \geq 2\varepsilon$

Proof. When E_3 occurs, the bit $b \in \{0,1\}$ indicating whether the challenge C^* is an encryption of w_0 or w_1 is independent of A's view. By definition of the security game, we know that $|\Pr[b = b'] - 1/2| \geq \varepsilon$. We next show that this implies that $\Pr[\neg\mathsf{E}_3] \geq 2\varepsilon]$ as follows:

$$\begin{aligned}\Pr[b = b'] &= \Pr[b = b' \mid \mathsf{E}_3]\Pr[\mathsf{E}_3] + \Pr[b = b' \mid \neg\mathsf{E}_3]\Pr[\neg\mathsf{E}_3] \\ &\leq \Pr[b = b' \mid \mathsf{E}_3]\Pr[\mathsf{E}_3] + \Pr[\neg\mathsf{E}_3] \\ &= \frac{1}{2}\Pr[\mathsf{E}_3] + \Pr[\neg\mathsf{E}_3] \\ &= \frac{1}{2} + \frac{1}{2}\Pr[\neg\mathsf{E}_3]\end{aligned}$$

$$\Pr[b = b'] \geq \Pr[b = b' \mid \mathsf{E}_3]\Pr[\mathsf{E}_3] = \frac{1}{2}\Pr[\mathsf{E}_3] = \frac{1}{2} - \frac{1}{2}\Pr[\neg\mathsf{E}_3]$$

It follows that $\varepsilon \leq |\Pr[b = b'] - 1/2| \leq \frac{1}{2}\Pr[\neg\mathsf{E}_3]$. Hence, $\Pr[\neg\mathsf{E}_3] \geq 2\varepsilon$.

By claim 3, if B does not abort during the simulation, B will choose a correct tuple in H_2-list with probability at least $1/q_{H_2}$, and will produce the correct answer with probability at least ε/q_{H_2}. Overall by combining Claim 1, 2, and 3, we have B's success probability is at least $\varepsilon/(eq_{td}q_{H_2})$.

4. Bidirectional Re-dPEKS Scheme

4.1. Definition

We define searchable proxy re-encryption scheme with a designated tester (Re-dPEKS) and only consider bidirectional, multi-use Re-dPEKS.

Definition 3. ***(Bidirectional, Multi-use Re-dPEKS)*** *A bidirectional, multi-use, searchable proxy re-encryption with a designated tester (Re-dPEKS) scheme consists of the following algorithms:*

- GlobalSetup(1^k): On input a security parameter 1^k, it returns a global parameter, GP.

- $\mathsf{KeyGen}_S(\mathsf{GP})$: On input GP, it returns a public-private key pair $[pk_S, sk_S]$ of the server S.
- $\mathsf{KeyGen}_R(\mathsf{GP})$: On input GP, it returns a public-private key pair $[pk_R, sk_R]$ of the receiver R.
- $\mathsf{ReKeyGen}(\mathsf{GP}, sk_{R_i}, sk_{R_j})$: On input GP, a private key sk_{R_i}, and a private key sk_{R_j}, where $i \neq j$, it returns a re-encryption key $rk_{R_i \leftrightarrow R_j}$ for receiver R_j
- $\mathsf{dPEKS}(\mathsf{GP}, pk_{R_i}, pk_S, w)$: On input GP, pk_{R_i}, pk_S, and a keyword w, it returns a dPEKS ciphertext $C_{i,w}$ of w.
- $\mathsf{RedPEKS}(\mathsf{GP}, rk_{R_i \leftrightarrow R_j}, C_{i,w})$: On input GP, $rk_{R_i \leftrightarrow R_j}$, and an original dPEKS ciphertext $C_{i,w}$, it returns a re-encryption dPEKS ciphertext $C_{j,w}$ of w for receiver R_j.
- $\mathsf{dTrapdoor}(\mathsf{GP}, pk_S, sk_{R_i}, w)$: On input GP, pk_S, sk_{R_i}, and a keyword w, it returns a trapdoor $T_{i,w}$.
- $\mathsf{dTest}(\mathsf{GP}, sk_S, C, T_w)$: On input GP, sk_S, a dPEKS ciphertext $C = \mathsf{dPEKS}(\mathsf{GP}, pk_R, pk_S, w')$, and a trapdoor $T_w = \mathsf{dTrapdoor}(\mathsf{GP}, pk_S, sk_R, w)$, it returns 1 if $w = w'$ and 0 otherwise.

Correctness: Let key pairs $[pk_{R_i}, sk_{R_i}]$ and $[pk_{R_j}, sk_{R_j}] \leftarrow KeyGen_R(\mathsf{GP})$, $rk_{R_i \leftrightarrow R_j} \leftarrow ReKeyGen(\mathsf{GP}, sk_{R_i}, sk_{R_j})$, $C_{i,w'} \leftarrow dPEKS(\mathsf{GP}, pk_{R_i}, pk_S, w')$, $T_{i,w} \leftarrow dTrapdoor(\mathsf{GP}, pk_S, sk_{R_i}, w)$, $T_{j,w} \leftarrow dTrapdoor(\mathsf{GP}, pk_S, sk_{R_j}, w)$, $\forall w, w' \in$ keyword space KW, it holds that

- $\mathsf{dTest}(\mathsf{GP}, sk_S, C_{i,w'}, T_{i,w}) = 1$ if $w = w'$, and 0 otherwise.
- $\mathsf{dTest}(\mathsf{GP}, sk_S, RedPEKS(\mathsf{GP}, rk_{R_i \leftrightarrow R_j}, C_{i,w'}), T_{j,w}) = 1$ if $w = w'$, and 0 otherwise.

We note that our searchable proxy re-encryption scheme with a designated tester (Re-dPEKS) is an extended searchable public key encryption scheme with a designated tester (dPEKS). In particular, we can add two algorithms, i.e., ReKeyGen and RedPEKS, in a dPEKS scheme to form a Re-dPEKS scheme.

4.2. Security Model

We need to consider the adversary is either a malicious server or a malicious user. A malicious server should not be able to distinguish which keyword corresponds to a given ciphertext without the trapdoor from the target receiver or a delegatee. A malicious user should not be able to distinguish which keyword corresponds to a target ciphertext without the server's secret key even s/he has the trapdoor of the keyword. We can model these two adversaries with two separate security games. Alternatively, we can use one game to simulate the capability of the adversaries by allowing them to call some restricted functions, such as $\mathsf{dTrapdoor}$ for malicious server and dTest for malicious user.

Re-dPEKS CKA-Security Game. The game consists of the following phases, which are executed in order. The oracles in each phase can be executed poly(k) times in any order unless otherwise specified. We assume a static corruption model: i.e., adversary has to determine either corrupt a party or not at the time the key pair of each party is generated.

1. Game Setup:
   - **Global Parameter Generation**: The challenger runs GlobalSetup(1^k) to generate the public parameter GP and gives it to the adversary A. This oracle is executed first and only once.
   - **Server Key Generation**: The challenger runs $\mathsf{KeyGen}_S(\mathsf{GP})$ to generate a public and private key pair $[pk_S, sk_S]$ and gives pk_S to A. This oracle is executed only once.
   - **Uncorrupted Receiver Key Generation**: The challenger runs $\mathsf{KeyGen}_R(\mathsf{GP})$ and returns a public-private key pair $[pk_R, sk_R]$. It gives pk_R to A. Let L_H be the set of honest receiver indices.
   - **Corrupted Receiver Key Generation**: The challenger runs $\mathrm{KeyGen}_R(\mathsf{GP})$ and returns a public-private key pair $[pk_R, sk_R]$. It gives $[pk_R, sk_R]$ to A. Let L_C be the set of corrupt receiver indices.
2. Phase 1: A makes the following queries:
   - **Trapdoor Generation** O_{td} : On input (i, w) by the adversary, where $i \in \mathsf{L}_H \cup \mathsf{L}_C$, $w \in$ keyword space KW , the challenger runs $\mathsf{dTrapdoor}(\mathsf{GP}, pk_S, sk_i, w)$ and returns a trapdoor $T_{i,w}$ associated with keyword w which generated by secret key of user i to A.
   - **Re-encryption Key Generation** O_{rk}: On input (i, j) by the adversary, where $i \neq j$, the challenger runs $\mathsf{ReKeyGen}(\mathsf{GP}, sk_i, sk_j)$ and returns a re-encryption key $rk_{i \leftrightarrow j}$ to A. We restrict that either both i and j are corrupted or both are uncorrupted, i.e., $i, j \in \mathsf{L}_H$ or $i, j \in \mathsf{L}_C$. In another words, re-encryption key queries between a corrupted and an uncorrupted party are not allowed.
   - **Re-encryption** O_{renc}: On input (i, j) and an original dPEKS ciphertext $C_{i,w}$ by the adversary, where i, j are both either from L_H or L_C, the challenger returns the re-encrypted dPEKS ciphertext $C_{j,w}$ = $\mathsf{RedPEKS}(\mathsf{GP}, \mathsf{ReKeyGen}(\mathsf{GP}, sk_i, sk_j), pk_S, C_{i,w})$. All re-encryption queries where $i = j$ or where $i \in \mathsf{L}_H$ and $j \in \mathsf{L}_C$ are ignored, i.e., an output of $\perp$.
   - **Test** O_{te} :On input (C, T_w) by the adversary, the challenger returns the output of $\mathsf{dTest}(\mathsf{GP}, sk_S, C, T_w)$.
3. Challenge Oracle: On input (i^*, w_0, w_1) by the adversary, where $w_0, w_1 \in \mathsf{KW}$, the challenger picks a random $b \in \{0,1\}$ and returns the challenge ciphertext $C^* = dPEKS(\mathsf{GP}, pk_{R_{i^*}}, pk_S, w_b)$ to A. The restriction is that $i^* \in \mathsf{L}_H$.

4. Phase 2: The adversary is allowed to ask the same types of queries as in Phase 1, except the following queries:
   - $\mathsf{O}_{td}(i^*, w_b)$ and $\mathsf{O}_{te}(C^*, T_{i^*,w_b})$, where $b \in \{0,1\}$.
   - $\mathsf{O}_{rk}(i^*, j)$ (or $\mathsf{O}_{rk}(j, i^*)$), $\mathsf{O}_{td}(j, w_b)$, and $\mathsf{O}_{te}(C_{j,w_b}, T_{j,w_b})$, where $b \in \{0,1\}$.
   - $\mathsf{O}_{renc}(i^*, j, C^*)$, $\mathsf{O}_{td}(j, w_b)$, and $\mathsf{O}_{te}(C_{j,w_b}, T_{j,w_b})$, where $b \in \{0,1\}$.

It is also restricted that the above list of queries should not be appeared in the whole simulation. For example, if A previously asked $\mathsf{O}_{td}(i^*, w_b)$ in phase 1, it is restricted to ask $\mathsf{O}_{te}(C^*, T_{i^*,w_b})$ in phase 2, where $b \in \{0,1\}$.

5. Guess: Finally, A outputs a guess $b' \in \{0,1\}$ and wins the game if $b = b'$.

We define A's advantage in breaking the Re-dPEKS scheme as:

$$Adv_{\mathsf{A}}^{\mathsf{IND-CKA\,Re-dPEKS}}(k) = |\Pr[b = b'] - 1/2|.$$

Definition 4. *We say that a Re-dPEKS scheme is indistinguishability against an adaptive chosen keyword attack (IND-CKA) if for any polynomial time attacker* A *we have that* $Adv_{\mathsf{A}}^{\mathsf{IND-CKA\,Re-dPEKS}}(k)$ *is a negligible function.*

4.3. Construction

We construct a secure bidirectional, multi-use Re-dPEKS scheme based on dPEKS scheme in [17, 16] and use the re-encryption technique in [9, 5].

- GlobalSetup(1^k): Let G and G_T be bilinear groups of order q. Let $H_1 : \{0,1\}^* \to \mathsf{G}$, and $H_2 : \mathsf{G}_T \to \{0,1\}^k$ be independent hash functions. Given a security parameter k, the algorithm picks a random generator $g \in \mathsf{G}$. It returns a global parameter $\mathsf{GP} = (q, \mathsf{G}, \mathsf{G}_T, g, e, H_1, H_2)$.
- KeyGen_S (GP): On input GP, select a random value $a \in \mathsf{Z}_q$. Set server's public key $pk_S = g^a$ and private key $sk_S = a$. Return $[pk_S, sk_S]$.
- KeyGen_R (GP): On input GP, select a random value $x \in \mathsf{Z}_q$. Set receiver's public key $pk_R = g^x$ and private key $sk_R = x$. Return $[pk_R, sk_R]$.
- ReKeyGen($\mathsf{GP}, sk_{R_X}, sk_{R_Y}$): On input $sk_{R_X} = x$ and $sk_{R_Y} = y$, output the bidirectional re-encryption key $rk_{R_X \leftrightarrow R_Y} = y/x \bmod q$.
- dPEKS(GP, pk_R, pk_S, w): To encrypt a keyword $w \in$ keyword space KW under receiver's public key $pk_R = g^x$ and designated server's public key $pk_S = g^a$, select a random value $r \in \mathsf{Z}_q$ and compute the ciphertext $C = [A, B] = [g^{xr}, H_2(e(g^a, H_1(w)^r))]$.

- RedPEKS(GP, $rk_{R_X \leftrightarrow R_Y}$, $C_{X,w}$): On input a re-encryption key $rk_{R_X \leftrightarrow R_Y}$, and a dPEKS ciphertext $C = [A, B]$, compute $A' = A^{rk_{R_X \leftrightarrow R_Y}} = g^{(xr)(y/x)} = g^{yr}$. Output the re-encrypted ciphertext from user R_X to R_Y as $C' = [A', B]$.
- dTrapdoor(GP, pk_S, sk_{R_X}, w): On input a receiver's private key sk_{R_X} and a keyword w, select a random $r' \in Z_q$ and output the trapdoor for a designated server S with public key $pk_S = g^a$ as $T_w = [U, V] = [g^{r'}, H_1(w)^{1/x} \cdot (g^a)^{r'}]$.
- dTest(GP, sk_S, C, T_w): On input a dPEKS ciphertext $C = [A, B]$ and a trapdoor $T_w = [U, V]$, the designated server with private key $sk_S = a$ first computes $T = V/U^a$ and then checks if $B = H_2(e(A, T^a))$. It outputs 1 if the above equality holds, and 0 otherwise.

Correctness:

Assume the dPEKS ciphertext of keyword w' is $[A, B] = [g^{xr}, H_2(e(g^a, H_1(w')^r))]$ and the trapdoor associated to keyword w is $[U, V] = [g^{r'}, H_1(w)^{\frac{1}{x}} \cdot (g^a)^{r'}]$, we have $T = \frac{H_1(w)^{\frac{1}{x}} \cdot (g^a) r'}{(g^{r'})^a} = H_1(w)^{\frac{1}{x}}$.

If $w = w'$, $B = H_2(e(g^a, H_1(w')^r)) = H_2(e(g^{xr}, H_1(w')^{\frac{a}{x}}) = H_2(e(A, T^a))$. It is easy to verify that the correctness of the equation holds for multi-test as re-encrypted ciphertext has the same form as the original ciphertext.

4.4. Security Analysis

Theorem 2. *The Re-dPEKS scheme is IND-CKA secure in the random oracle model assuming that the mBDH problem is intractable.*

Proof. Let A be a polynomial-time attack algorithm. We construct an algorithm B that solve the mBDH problem in G, G_T. On mBDH input $(g, u_1 = g^\alpha, u_2 = g^\beta, u_3 = g^\gamma \in G)$, B's goal is to compute $e(g, g)^{\alpha\beta/\gamma}$. B simulates the challenger and interacts with A as follows:

1. Game Setup:

- **Public Parameter Generation**: B setups the global parameter GP and gives $GP = (q, G, G_T, g, e, H_1, H_2)$ to A.
- **Server Key Generation**: On input a designated server S, B selects a random $a \in Z_q$, and generates the public key $pk_S = g^a$ and private key $sk_S = a$. B gives pk_S to A.
- **Uncorrupted Receiver Key Generation**: On input an index i, B selects a random $x_i \in Z_q$, and outputs the public key $pk_{R_i} = u_3^{x_i} = (g^\gamma)^{x_i}$, where the private key is implicitly defined as $sk_{R_i} = \gamma x_i$. It adds the tuple $\langle i, pk_{R_i}, x_i \rangle$ in L_H.

- **Corrupted Receiver Key Generation**: On input an index i, B selects a random $x_i \in \mathsf{Z}_q$, and outputs $pk_{R_i} = g^{x_i}$ and $sk_{R_i} = x_i$. It adds the tuple $\langle i, pk_{R_i}, x_i \rangle$ in L_C.

2. Hash Function Queries: A can query the random oracle H_1 or H_2 at any time.

- H_1 **-query** O_{H_1} : B maintains an H_1 -list with tuples $\langle w_n, h_n, d_n, c_n \rangle$ which is initially empty. On input w_i, B responds as follows:
  - If the query w_i is found in the H_1 -list with an entry $\langle w_i, h_i, d_i, c_i \rangle$, output $H_1(w_i) = h_i$.
  - Otherwise, B selects a random $d_i \in \mathsf{Z}_q$ and generates a random coin $c_i \in \{0,1\}$ so that $\Pr[c_i = 0] = 1/(q_{td} + 1)$.
    - If $c_i = 1$, B computes $h_i = u_3^{d_i} = (g^{\gamma})^{d_i}$.
    - If $c_i = 0$, B computes $h_i = u_1^{d_i} = (g^{\alpha})^{d_i}$.

    B adds the tuple $\langle w_i, h_i, d_i, c_i \rangle$ to the H_1-list and returns $H_1(w_i) = h_i$.
- H_2 **-query** O_{H_2} : Similarly, B maintains an H_2 -list with tuples $\langle t, M \rangle$ which is initially empty. On input $t \in \mathsf{G}_T$, B responds with $H_2(t) = M$. For each new t, B responds to the $H_2(t)$ query by selecting a new random value $M \in \{0,1\}^k$ and setting $H_2(t) = M$. B then adds the tuple $\langle t, M \rangle$ to the H_2 list.

3. Phase 1: When A issues the following query, B responds as follows:

- **Trapdoor Generation**: On input (i, w_i) to O_{td}, do:
  - B gets the response from O_{H_1} to obtain an entry $\langle w_i, h_i, d_i, c_i \rangle$ in H_1-list. If $c_i = 0$, output $\perp$ and abort. Otherwise, we know $c_i = 1$ and we have $h_i = u_3^{d_i} = (g^{\gamma})^{d_i} \in \mathsf{G}$ in the H_1-list.
  - If $i \in \mathsf{L}_C$, B selects a random value $r' \in \mathsf{Z}_q$, and sets $U = g^{r'}$ and $V = g^{\frac{\gamma \cdot d_i}{x_i}} \cdot g^{ar'}$, where x_i is obtained from the tuple $\langle i, pk_{R_i}, x_i \rangle$ in L_C.
  - If $i \in \mathsf{L}_H$, B selects a random value $r' \in \mathsf{Z}_q$, and sets $U = g^{r'}$ and $V = g^{\frac{d_i}{x_i}} \cdot g^{ar'}$, where x_i is obtained from the tuple $\langle i, pk_{R_i}, x_i \rangle$ in L_H. Observe that $H_1(w_i)^{\frac{1}{\gamma x_i}} = (g^{\gamma \cdot d_i})^{\frac{1}{\gamma x_i}} = g^{\frac{d_i}{x_i}}$ and therefore V is the correct trapdoor component for the keyword w_i under the implicitly defined user's private key γx_i. B gives $T_w = [U, V]$ to A.
- **Re-encryption Key Generation**: On input (i, j) to O_{rk}, do:
  - If (1) $i \in \mathsf{L}_H$ and $j \in \mathsf{L}_C$ or (2) $i \in \mathsf{L}_C$ and $j \in \mathsf{L}_H$ or (3) $i = j$ or (4) i or j not in L_H or L_C, output $\perp$.

– Otherwise, output $rk_{i\leftrightarrow j} = x_j/x_i$.

- **Re-encryption** O_{renc}: On input $(i, j, C_{i,w})$ by the adversary where $C_{i,w}$ is an original dPEKS ciphertext, do:
  – If (1) $i \in \mathsf{L}_H$ and $j \in \mathsf{L}_C$ or (2) $i \in \mathsf{L}_C$ and $j \in \mathsf{L}_H$ or (3) $i = j$ or (4) i or j not in L_H or L_C, output $\perp$.
  – If i and j are both from L_H or they are both from L_C, B obtains the re-encryption key $rk_{i\leftrightarrow j} = x_j/x_i$ from O_{rk} and returns the re-encryption ciphertext $C_{j,w} = \mathsf{RedPEKS}(\mathsf{GP}, x_j/x_i, pk_S, C_{i,w})$ to A.
- **dTest** O_{te}: On input a dPEKS ciphertext $C = [A, B]$ and a trapdoor $T_w = [U, V]$, B first computes $\mathsf{T} = V/U^a$ and then sets $t = e(A, \mathsf{T}^a)$. B gets the response from O_{H_2} to obtain an entry $\langle t = t_i, M_i \rangle$ in H_2-list. B outputs 1 if $B = M_i$ and 0 otherwise.

4. Challenge Oracle: At some point, A gives the challenge tuple (i^*, w_0, w_1) to B, where $w_0, w_1 \in \mathsf{KW}$, do:
   – If (1) i^* is not from L_H, B returns $\perp$.
   – Otherwise, B asks O_{H_1} to obtain h_0, $h_1 \in \mathsf{G}$ such that $H_1(w_0) = h_0$ and $H_1(w_1) = h_1$. Let $\langle w_b, h_b, d_b, c_b \rangle$, where $b \in \{0,1\}$, be the corresponding tuples in H_1-list. If $c_0 = 1$ or $c_1 = 1$, B returns $\perp$ and aborts.
   – Otherwise, at least one of c_0 or c_1 is equal to 0. B picks a random $b \in \{0,1\}$ such that $c_b = 0$.
   – B selects a random $M^* \in \{0,1\}^k$ and sets $A^* = u_2^{x_{i^*}} = g^{\beta \cdot x_{i^*}}$ and $B^* = M^*$. This challenge implicitly defines $A^* = (pk_{R_{i^*}})^{\beta/\gamma} = (g^{\gamma \cdot x_{i^*}})^{\beta/\gamma}$. Also, it defines $M^* = H_2(e(g^a, H_1(w_b))^{\beta/\gamma}) = H_2(e(g^a, g^{\alpha \cdot d_b})^{\beta/\gamma}) = H_2(e(g,g)^{(\frac{\alpha\beta}{\gamma})(a \cdot d_b)})$. B returns the challenge ciphertext $C^* = [A^*, B^*]$ to A.
5. Phase 2: A is allowed to ask the same types of queries as in Phase 1 and B responds identically in Phase 1, except the following queries where B returns $\perp$:
   $\mathsf{O}_{td}(i^*, w_b)$ and $\mathsf{O}_{te}(C^*, T_{i^*, w_b})$, where $b \in \{0,1\}$.
   – $\mathsf{O}_{rk}(i^*, j)$ (or $\mathsf{O}_{rk}(j, i^*)$), $\mathsf{O}_{td}(j, w_b)$, and $\mathsf{O}_{te}(C_{j,w_b}, T_{j,w_b})$, where $b \in \{0,1\}$.
   – $\mathsf{O}_{renc}(i^*, j, C^*)$, $\mathsf{O}_{td}(j, w_b)$, and $\mathsf{O}_{te}(C_{j,w_b}, T_{j,w_b})$, where $b \in \{0,1\}$.

   It is also restricted that the above list of queries should not be appeared in the whole simulation.
6. Guess: Finally, A outputs a guess $b' \in \{0,1\}$. B picks a random pair $\langle t, M \rangle$ from the H_2-list and outputs t^{1/ad_b} as its guess for $e(g,g)^{\alpha\beta/\gamma}$, where d_b is the value used in the Challenge step.

Since A must have asked a query of either $H_2(e(g^a, H_1(w_0)^{\beta/\gamma}))$ or $H_2(e(g^a, H_1(w_1)^{\beta/\gamma}))$, the H_2 -list contains a tuple where $t = e(g^a, H_1(w_b)^{\beta/\gamma}) = e(g,g)^{(\alpha\beta/\gamma)(ad_b)}$ with probability 1/2. If B picks this tuple $\langle t, M \rangle$, then $t^{1/ad_b} = e(g,g)^{\alpha\beta/\gamma}$ as required.

5. Conclusions

We proposed new definitions for searchable proxy re-encryption scheme (Re-PEKS) and searchable proxy re-encryption scheme with a designated tester (Re-dPEKS). We gave a construction for each scheme and proved the security in the random oracle model.

There are some interesting open problems as follows:

- Find efficient Re-(d)PEKS schemes secure in the standard model.
- Construct secure unidirectional Re-(d)PEKS schemes.
- Explore the secure integration of PRE and Re-(d)PEKS, such as previous works on combining PKE and PEKS in [3, 22].
- Find efficient Re-(d)PEKS schemes secure against keyword guessing attacks [8,20,13].

References

[1] Abdalla, M., Bellare, M., Catalano, D., Kiltz, E., Kohno, T., Lange, T., Malone-Lee, J., Neven, G., Paillier, P., Shi, H.: Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions. In: Shoup, V. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 3621, pp. 205-222. Springer (2005)

[2] Ateniese, G., Fu, K., Green, M., Hohenberger, S.: Improved proxy re-encryption schemes with applications to secure distributed storage. ACM Trans. Inf. Syst. Secur. 9(1), 1-30 (2006)

[3] Baek, J., Safavi-Naini, R., Susilo, W.: On the integration of public key data encryption and public key encryption with keyword search. In: Katsikas, S.K., Lopez, J., Backes, M., Gritzalis, S., Preneel, B. (eds.) ISC. Lecture Notes in Computer Science, vol. 4176, pp. 217-232. Springer (2006)

[4] Baek, J., Safavi-Naini, R., Susilo, W.: Public key encryption with keyword search revisited. In: Gervasi, O., Murgante, B., Lagan a, A., Taniar, D., Mun, Y., Gavrilova, M.L. (eds.) ICCSA (1). Lecture Notes in Computer Science, vol. 5072, pp. 1249-1259. Springer (2008)

[5] Blaze, M., Bleumer, G., Strauss, M.: Divertible protocols and atomic proxy cryptography. In: EUROCRYPT. pp. 127-144 (1998)

[6] Boneh, D., Crescenzo, G.D., Ostrovsky, R., Persiano, G.: Public key encryption with keyword search. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT. Lecture Notes in Computer Science, vol. 3027, pp. 506-522. Springer (2004)

[7] Boneh, D., Franklin, M.K.: Identity-based encryption from the weil pairing. In: Kilian, J. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 2139, pp. 213-229. Springer (2001)

[8] Byun, J.W., Rhee, H.S., Park, H.A., Lee, D.H.: Off-line keyword guessing attacks on recent keyword search schemes over encrypted data. In: Jonker, W., Petkovic, M. (eds.) Secure Data Management. Lecture Notes in Computer Science, vol. 4165, pp. 75-83. Springer (2006)

[9] Canetti, R., Hohenberger, S.: Chosen-ciphertext secure proxy re-encryption. In: Ning, P., di Vimercati, S.D.C., Syverson, P.F. ACM Conference on Computer and Communications Security. pp. 185-194. ACM (2007)

[10] Crescenzo, G.D., Saraswat, V.: Public key encryption with searchable keywords based on jacobi symbols. In: Srinathan, K., Rangan, C.P., Yung, M. (eds.) INDOCRYPT. Lecture Notes in Computer Science, vol. 4859, pp. 282-296. Springer (2007)

[11] Gu, C., Zhu, Y., Pan, H.: Efficient public key encryption with keyword search schemes from pairings. In: Pei, D., Yung, M., Lin, D., Wu, C. (eds.) Inscrypt. Lecture Notes in Computer Science, vol. 4990, pp. 372-383. Springer (2007)

[12] Hwang, Y.H., Lee, P.J.: Public key encryption with conjunctive keyword search and its extension to a multi-user system. In: Takagi, T., Okamoto, T., Okamoto, E., Okamoto, T. (eds.) Pairing. Lecture Notes in Computer Science, vol. 4575, pp. 2-22. Springer (2007)

[13] Jeong, I.R., Kwon, J.O., Hong, D., Lee, D.H.: Constructing peks schemes secure against keyword guessing attacks is possible? Computer Communications 32(2), 394-396 (2009)

[14] Park, D.J., Kim, K., Lee, P.J.: Public key encryption with conjunctive field keyword search. In: Lim, C.H., Yung, M. (eds.) WISA. Lecture Notes in Computer Science, vol. 3325, pp. 73-86. Springer (2004)

[15] Rhee, H.S., Park, J.H., Susilo, W., Lee, D.H.: Improved searchable public key encryption with designated tester. In: Li, W., Susilo, W., Tupakula, U.K., Safavi-Naini, R., Varadharajan, V. (eds.) ASIACCS. pp. 376-379. ACM (2009)

[16] Rhee, H.S., Park, J.H., Susilo, W., Lee, D.H.: Trapdoor security in a searchable public-key encryption scheme with a designated tester. Journal of Systems and Software 83(5), 763-771 (2010)

[17] Rhee, H.S., Susilo, W., Kim, H.J.: Secure searchable public key encryption scheme against keyword guessing attacks. IEICE Electronics Express 6(5), 237-243 (2009)

[18] Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: Cramer, R. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 3494, pp. 457-473. Springer (2005)

[19] Shao, J., Cao, Z., Liang, X., Lin, H.: Proxy re-encryption with keyword search. Inf. Sci. 180(13), 2576-2587 (2010)

[20] Yau, W.C., Heng, S.H., Goi, B.M.: Off-line keyword guessing attacks on recent public key encryption with keyword search schemes. In: Rong, C., Jaatun, M.G., Sandnes, F.E., Yang, L.T., Ma, J. (eds.) ATC. Lecture Notes in Computer Science, vol. 5060, pp. 100-105. Springer (2008)

[21] Yau, W.C., Phan, R.C.W., Heng, S.H., Goi, B.M.: Proxy re-encryption with keyword search: new definitions and algorithms.In: Kim, T.H., Fang, W.C., Khan, M.K., Arnett, K.P., Kang, H.J., Ślęzak, D. SecTech/DRBC. Communications in Computer and Information Science, vol. 122, pp. 149-160. Springer (2010)

[22] Zhang, R., Imai, H.: Generic combination of public key encryption with keyword search and public key encryption. In: Bao, F., Ling, S., Okamoto, T., Wang, H., Xing, C. (eds.) CANS. Lecture Notes in Computer Science, vol. 4856, pp. 159-174. Springer (2007)

Authors

Wei-Chuen Yau is a lecturer in the Faculty of Engineering at Multimedia University, Malaysia. He received his B.Sc and M.Sc degrees from National Cheng Kung University in 1999 and 2001, respectively. He is currently pursuing his PhD study at Multimedia University. His research interests include cryptography and network security.

Raphael Phan holds a PhD from Multimedia University. Prior to joining Loughborough University, Raphael was Director of the Information Security Research (iSECURES) Laboratory at Swinburne Uni of Tech; and then a researcher in the Security & Cryptography Lab (LASEC) at the Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland. He passionately researches on security and privacy, in ciphers, protocols, networks, systems; and more generally anything to do with malice. He is in the Editorial Board of Cryptologia, and the Cryptology & Information Security series of IOS Press. He is General Chair of Mycrypt '05 and Asiacrypt '07, Program Chair of ISH '05, and serves in technical Program Committees of international conferences since 2005.

Raphael is co-designer of BLAKE, one of the five hash functions in the final of the NIST SHA-3 competition.

Swee-Huay Heng received her B.Sc (Hons) and M.Sc degrees from University Putra Malaysia (UPM), and her Doctor of Engineering degree from the Tokyo Institute of Technology, Japan. She is currently an Associate Professor in the Faculty of Information Science & Technology, Multimedia University, Malaysia. Her research interests include Cryptography and Information Security. She was the Program Chair of ProvSec 2010 and CANS 2010. She has been actively involved in technical Program Committees of several international security conferences.

Bok-Min Goi received his B.Eng degree from University of Malaya (UM) in 1998, and the M.Eng.Sc and Ph.D degrees from Multimedia University (MMU) in 2002 and 2006, respectively. He is now the Deputy Dean (R&D and Postgraduate Programmes) and an associate professor in the Faculty of Engineering and Science, Universiti Tunku Abdul Rahman (UTAR), Malaysia. His research interests include cryptology, security protocols, information security, digital watermarking and embedded systems design.