An enhanced searchable public key encryption scheme with a designated tester and its extensions

Chengyu Hu  
School of Computer Science and Technology, Shandong University, Jinan, P.R.China  
Email: hcy@mail.sdu.edu.cn

Pengtao Liu  
Institute of Information Science and Technology, Shandong University of Political Science and Law, Jinan, P.R.China  
Email: ptwave@163.com

Abstract—In a searchable public-key encryption scheme with a designated tester (dPEKS), only the designated server can test which dPEKS ciphertext is related with a given trapdoor generated by a user with a keyword w by using the server’s private key, but learn nothing else. In this paper, we study the keyword guessing attack of dPEKS and shows that two dPEKS schemes are insecure against this type of attack. Then an enhanced scheme is proposed and proved secure against chosen keyword attack and keyword guessing attack. To resolve the problem that dPEKS scheme does not allow the user to decrypt the encrypted keyword which limits its applicability, we give the definition of decryptable searchable public key encryption with a designated tester which enables keyword decryption from keyword ciphertext and get a concrete construction based on our dPEKS scheme. At last, we extend the dPEKS scheme to a bidirectional searchable proxy re-encryption with a designated tester scheme (Re-dPEKS).

Index Terms—searchable public key encryption, designated tester, keyword guessing attack, decryptable, re-dpeks

I. INTRODUCTION

With the development of cloud computing services, more and more sensitive data are being stored and managed on the cloud servers. To ensure data privacy and confidentiality from even inside attackers such as a malicious cloud service provider, sensitive data has to be encrypted before outsourcing. However, data encryption transforms data into random strings and makes effective data utilization a very challenging task. For example, a user might want to only retrieve certain specific data he is interested in. Keyword-based search technique is one of the most popular ways to solve the retrieve problem and has been widely applied in plaintext search scenarios. Unfortunately, traditional plaintext search methods can not be applied in encrypted data because the server cannot read the data information so that it cannot answer any search queries and the user demands the protection of keyword privacy.

To resolve this problem, in 2004, Boneh et al.\cite{1} proposed the concept of public key encryption with keyword search scheme (PEKS) to enable a receiver to search encrypted keywords under his public key without compromising the security of the original data and proposed a universal transformation from anonymous identity-based encryption (IBE)\cite{2, 3, 4} to PEKS. PEKS is also referred to as searchable encryption\cite{1}. In a public key encryption with keyword search scheme, a sender sets a list of keywords and encrypts data using a standard encryption. The user then appends to the resulting ciphertext a PEKS ciphertext of the selected keyword. This kind of encrypted data may be stored in a server. To search the encrypted data with a specific keyword on the server, the user can give the server a certain trapdoor \( T_w \) of the searched keyword that enables the server to test whether the keyword associated with the data is identical to the keyword of the trapdoor without revealing any information about the searched keyword. Abdalla et al. presented an improved universal transformation from anonymous IBE to PEKS and three extensions, namely anonymous Hierarchical IBE (HIBE), public-key encryption with temporary keyword search (PETKS), and identity-based encryption with keyword search \cite{5}. To achieve conjunctive keyword search, two public key encryption schemes with conjunctive keyword search (PECKS)\cite{6, 7} were respectively proposed. In [8], Baek et al. pointed out that in Boneh et al.’s PEKS the linkability of encrypted data can be revealed to anyone for a given trapdoor so that it uses a secure channel between the receiver and the server, which is usually costly. To remove the secure channel, Baek et al. also proposed a searchable public key encryption for a designated tester (dPEKS)\cite{8}, in which only the designated server can test whether or not a given dPEKS ciphertext is related with a trapdoor by using its private key. Most of the proposed PEKS schemes prove their security in the condition that the size of keyword space must be beyond the polynomial level which is obviously unreasonable in practice. Therefore, under the practical condition that the size of keyword space is not more than the polynomial level, the security definition for PEKS does not capture the vulnerability of an off-line keyword guessing attack or...
dictionary attack by a curious server, which can access the trapdoors of limited keywords. Byun et al. firstly defined off-line keyword guessing (KG) attacks[9] which enables the adversary to learn which keyword is used to generate the trapdoor and show that the PEKS scheme in [1] is insecure against KG attacks. Since a user usually queries commonly-used keywords with low entropy, the keyword guessing attacks are meaningful. If an attacker who can guess the keyword \( w \) of the given trapdoor \( T_w \) obtains PEKS/dPEKS ciphertexts, it can know that not only the relation between the PEKS/dPEKS ciphertexts and the trapdoor \( T_w \), but also the keyword about PEKS/dPEKS ciphertexts by using KG attacks. Jeong et al. proved that any PEKS scheme satisfying at least computationally indistinguishable consistency implies successful keyword guessing attacks\[10\]. However, it is possible to construct a secure dPEKS scheme against KG attacks\[11,12\].

Suppose Bob shares encrypted data to Alice in the cloud storage. Alice uses different electronic devices to get the data, and may download data to her devices depending on the associated keyword. For example, she may like to download data with the keyword “urgent” on her mobile phone, and all other data on her desktop computer. However, in some situations, Alice may want to sort the received encrypted data in her desktop computer using keywords without decrypting them. All of the schemes referred above do not allow to retrieve the keyword, also they do not guarantee any relation between PEKS/dPEKS ciphertext and keyword, so that they are not applicable in this scenario. Decryptable searchable public key encryption which extends the notion of PEKS and enables decryption of keyword can resolve this problem. Fuhr and Paillier\[14\] put forward a construction for searchable public key encryption with decryption (PEKSD) scheme. However, it is improper that the scheme’s test method can decrypt the ciphertext and get the associated keyword. Fang et al. presented a decryptable searchable encryption scheme without random oracle\[15\]. But it is not the scheme with a designated tester.

For the scenario that user Alice may want to delegate her decryption right to another user Bob, the cloud server can use proxy re-encryption (PRE)\[16\] to convert encrypted data for Alice into ciphertext which can be decrypted by Bob. Moreover, it must be satisfied that during the conversion, the cloud server should not learn secret keys of Alice or Bob and the plaintext. Since the original keyword ciphertext encrypted under Alice’s public key cannot be tested with the trapdoor generated by user Bob, it is necessary to enable the server to search for those re-encrypted messages associated to certain keywords with the trapdoor \( T_w \) generated by user Bob. To do this, the server can perform the re-encryption of data ciphertext with a proxy re-encryption scheme, and re-encryption of keyword ciphertext using a proxy re-encryption with keyword search scheme (Re-dPEKS)\[17\] or searchable proxy re-encryption with a designated tester.

This paper analyzes the two searchable public key encryption schemes with a designated tester in [11,12] and shows that they are insecure against keyword guessing attacks. Then an enhanced scheme is constructed. We prove that the enhanced scheme is secure against chosen keyword attacks and keyword guessing attacks. We provide the definition of decryptable searchable encryption with a designated tester. At last, we extend our dPEKS scheme to a secure decryptable searchable encryption with a designated tester and a searchable proxy re-encryption with a designated tester. The rest of this paper is organized as follows. In Section 2, we review some preliminaries. In Section 3, we analyze two searchable encryption schemes with a designated tester in [11,12]. In Section 4, we present our new dPEKS scheme against keyword guessing attacks. The security analysis is given in Section 5. In Section 6, we provide the definition of decryptable searchable encryption with a designated tester and give a concrete construction based on our dPEKS scheme. Then, we extend the dPEKS scheme to a bidirectional searchable proxy re-encryption with a designated tester scheme (Re-dPEKS). Finally, we draw our conclusions in Section 7.

II. PRELIMINARIES

We briefly describe mathematical background and complexity assumptions used throughout this paper and the definition of dPEKS, dPEKSD and Re-dPEKS.

A. Bilinear Pairings

Let \( G_1 \) be a cyclic group generated by \( g \), with a prime order \( p \), and \( G_2 \) be a cyclic group with the same prime order \( p \). Let \( e : G_1 \times G_1 \rightarrow G_2 \) be a map with the following properties\[18\]:

1) Bilinearity: \( e(g^a, g^b) = e(g, g)^{ab} \) for any \( a, b \in \mathbb{Z}_p^* \);
2) Non-degeneracy: \( e(g, g) \) is a generator of \( G_2 \) which is also denoted by \( g_2 \), i.e., \( g_2 \neq 1_{G_2} \);
3) Computability: There is an efficient algorithm to compute \( e(u, v) \) for all \( u, v \in G_1 \);

Examples of cryptographic bilinear maps include modified Weil pairing\[2\] and Tate pairing\[19\]: \( G_1 \) is a group of points on an elliptic curve and \( G_2 \) is a multiplicative subgroup of a finite field. The traditional notation for group \( G_1 \) originates from elliptic curve groups and thus is additive. However we prefer a multiplicative notation for simplicity.

The Discrete Logarithm (DLP) problem is the problem of finding the least positive integer \( a \) such that the equation \( h = g^a \) holds, when the elements \( g, h \in G_1 \) are given, provided this integer exists.

The Hash Diffie-Hellman (HDH) assumption\[20\]: Let \( hLen \) be a number and \( H : \{0, 1\}^* \rightarrow \{0, 1\}^{hLen} \) be a hash function. The HDH problem in \( G_1 \) is defined as follows:

Given \( (g, g^a, g^b, H(g^c)) \) as inputs, output "yes" if \( ab = c \) and "no" otherwise. An algorithm \( A \) that outputs \( b' \in 0, 1 \) has an advantage \( \epsilon \) in solving the HDH problem in \( G_1 \) if

\[
|Pr[A(g, g^a, g^b, H(g^c)) = "yes"] - Pr[A(g, a, b, Z_p)]| = \Omega(\epsilon)
\]
where the probability is taken over the random choice of \( g \in G_1 \), the random choice of \( \eta \in \{0, 1\}^{h \cdot \text{Len}} \), the random choice of \( a, b \in Z_p \), and the random bits of \( A \). We say that the HDH assumption holds in \( G_1 \) if no \( t \)-time algorithm has an advantage at least \( \epsilon \) in solving the HDH problem in \( G_1 \).

### B. Searchable public-key encryption scheme with a designated tester

A searchable public-key encryption scheme with a designated tester (dPEKS) involves the following entities: senders, a receiver, and a server.

**Searchable public-key encryption scheme with a designated tester (dPEKS).** A searchable public-key encryption scheme with a designated tester consists of the following polynomial time algorithms where \( gp \) denotes a set of global parameters:

1. **GlobalSetup(\( \lambda \)):** This algorithm takes as input a security parameter \( \lambda \) and generates a global parameter \( gp \).
2. **KeyGenServer\((gp)\):** This algorithm takes input \( gp \), outputs a pair of public and secret keys \((pk_S, sk_S)\), of server \( S \).
3. **KeyGenReceiver\((gp)\):** This algorithm takes input \( gp \), outputs a pair of public and secret keys \((pk_R, sk_R)\), of receiver \( R \).
4. **dTrapdoor\((gp, pk_S, sk_R, w)\):** This algorithm takes as input \( gp \), the server’s public key, \( pk_S \), the receiver’s secret key, \( sk_R \), and a keyword, \( w \). It then generates a trapdoor, \( T_w \).
5. **dPEKS\((gp, pk_R, pk_S, w)\):** This algorithm takes as input \( gp \), the receiver’s public key, \( pk_R \), the server’s public key, \( pk_S \), and a keyword, \( w \). It returns a dPEKS ciphertext, \( C \) of \( w \).
6. **dTest\((gp, C, sk_S, T_w)\):** This algorithm takes as input \( gp \), a dPEKS ciphertext, \( C \), the server’s secret key, \( sk_S \), and a trapdoor, \( T_w \). It outputs ‘yes’ if \( w = w' \) and ‘no’ otherwise, where \( C \) = dPEKS\((gp, pk_R, pk_S, w')\).

With a dPEKS scheme, a sender encrypts her message, runs dPEKS to generate some keyword ciphertext \( C \) for the message, and stores the encrypted message and the keyword ciphertext at the server. The receiver then runs dTrapdoor to generate a trapdoor for selected keyword, and sends the trapdoor to the server which will run dTest to search for the keyword ciphertext attached to each encrypted message[21].

**Security Model of dPEKS.** Let \( A_i \) (\( i = 1, 2 \)) be an adversary whose running time is bounded by \( t \) which is polynomial in a security parameter \( k \). Similar to [12], we consider the following two games between the attacker \( A_1 \) (or \( A_2 \)) and a challenger \( B \):

- **Game 1.** \( A_1 \) is assumed to be a malicious server (insider attacker).
  
  **Setup:** \( A_1 \) generates the pair of his public/secret keys \((pk_S, sk_S)\) and gives \( pk_S = pk_A \) to \( B \). \( B \) generates the receiver’s pair of public/secret keys \((pk_R, sk_R)\) and gives \( pk_R \) to \( A_1 \). Here, \((pk_S, sk_S)\) and \( pk_R \) are given to \( A_1 \) and \( pk_S \) and \((pk_R, sk_R)\) are given to \( B \).
  
  **Phase 1 (dTrapdoor queries):** \( A_1 \) can adaptively asks \( B \) for the trapdoor \( T_w \) for any keyword \( w \) of his choice. And \( A_1 \) can get the associated keyword \( w \) about \( C \). Also, \( A_1 \) can get the test result about \( C \) and the given \( T_w \). To get the trapdoor \( T_w \) for any keyword \( w \) of his choice, \( A_1 \) makes the dTrapdoor query.
  
  **Challenge:** \( A_1 \) gives \( pk_R, w_0, \) and \( w_1 \) to \( B \). The restriction is that \( A_1 \) did not previously ask for the trapdoors \( T_{w_0} \) or \( T_{w_1} \), \( B \) chooses a random \( b \in \{0, 1\} \) and computes \( C^* = \text{dPEKS}(gp, pk_R, pk_S, w_b) \), and sends \( C^* \) to \( A_1 \).
  
  **Game 2.** \( A_2 \) is assumed to be an outsider attacker.
  
  **Setup:** \( B \) generates the server’s pair of public/secret keys \((pk_S, sk_S)\) and the receiver’s public/secret keys \((pk_R, sk_R)\) and gives \( pk_S, pk_R \) to \( A_2 \).
  
  **Phase 1 (dTrapdoor and dTest queries):** \( A_2 \) can adaptively asks \( B \) for the test result about \( C \) and the given \( T_w \) of his choice as long as \( w \neq w_0, w_1 \).
  
  **Guess:** \( A_2 \) outputs \( b' \in \{0, 1\} \) and wins Game 2 if \( b = b' \).
  
  We define \( A_1 \)’s advantage in breaking the dPEKS as \( \text{Adv}_{A_1}(\lambda) = |Pr[b = b'] - 1/2| \).

- **Game 2.** \( A_2 \) is assumed to be an outsider attacker.
  
  **Setup:** \( B \) generates the server’s pair of public/secret keys \((pk_S, sk_S)\) and the receiver’s public/secret keys \((pk_R, sk_R)\) and gives \( pk_S, pk_R \) to \( A_2 \).
  
  **Phase 1 (dTrapdoor and dTest queries):** \( A_2 \) can adaptively asks \( B \) for the test result about \( C \) and the given \( T_w \) of his choice. Also, \( A_2 \) can issue a query for the dTrapdoor that corresponds to the keyword \( w \).
  
  **Challenge:** \( A_2 \) gives \( w_0, \) and \( w_1 \) to \( B \). The restriction is that \( A_2 \) did not previously ask for the trapdoors \( T_{w_0} \) or \( T_{w_1} \) to dTest query. \( B \) chooses a random \( b \in \{0, 1\} \) and computes \( T_w^* = \text{dTrapdoor}(gp, pk_R, sk_R, w_b) \) and sends \( T_w^* \) to \( A_2 \).
  
  **Phase 2 (dTrapdoor and dTest queries):** This is identical to Phase 1, except that \( A_2 \) may not issue the dTrapdoor query for \( T_w \) and test query for \((C, sk_S, T_w)\) where the corresponding elements of \( C \) is not same to one of \( C^* \) and trapdoor \( T_w \) for any keyword \( w \) of his choice as long as \( w \neq w_0, w_1 \).
  
  **Guess:** \( A_2 \) outputs \( b' \in \{0, 1\} \) and wins Game 2 if \( b = b' \).
  
  We define \( A_2 \)’s advantage in breaking the dPEKS as \( \text{Adv}_{A_2}(\lambda) = |Pr[b = b'] - 1/2| \).

We say that a dPEKS is secure against an adaptive chosen keyword attack if for any polynomial time attackers \( A_i \) (\( i = 1, 2 \)) we have that \( \text{Adv}_{A_i}(\lambda) \) is negligible.

(Parital)off-line keyword guessing attack[21]. As is shown in [17], in many application scenarios of PEKS, it is reasonable to assume that the keyword set is public and has polynomial size in the security parameter. For
example referred in [21], in the email routing case, we could expect the size of the keyword set Urgent, Normal, to be very small. For a PEKS scheme, the server can generate a keyword ciphertext for each keyword and test it with the trapdoors at hand. Therefore, it may sort out the relationships between keywords and the trapdoors that it has received. With the knowledge of the relationships between keywords and the trapdoors, given a new keyword ciphertext, the server can determine the embedded keyword and therefore violate the privacy. Even if the word ciphertext, the server can determine the embedded keyword. With the knowledge of the relationships between keywords and the trapdoors that it has received. With the knowledge of the relationships between keywords and the trapdoors, given a new keyword ciphertext, the server can determine the embedded keyword and therefore violate the privacy. Even if the word ciphertext, the server can determine the embedded keyword. With the knowledge of the relationships between keywords and the trapdoors, given a new keyword ciphertext, the server can determine the embedded keyword and therefore violate the privacy. Even if the word ciphertext, the server can determine the embedded keyword. With the knowledge of the relationships between keywords and the trapdoors, given a new keyword ciphertext, the server can determine the embedded keyword and therefore violate the privacy. Even if the word ciphertext, the server can determine the embedded keyword. With the knowledge of the relationships between keywords and the trapdoors, given a new keyword ciphertext, the server can determine the embedded keyword and therefore violate the privacy. Even if the word ciphertext, the server can determine the embedded keyword. With the knowledge of the relationships between keywords and the trapdoors, given a new keyword ciphertext, the server can determine the embedded keyword and therefore violate the privacy. Even if the word ciphertext, the server can determine the embedded keyword. With the knowledge of the relationships between keywords and the trapdoors, given a new keyword ciphertext, the server can determine the embedded key.

C. Bidirectional searchable proxy re-encryption with a designated tester

In [22], Shao et al. provided the definition of proxy re-encryption with keyword search (PRES) which includes encryption and decryption of message. Their scheme encrypts the message and keyword in the same encryption algorithm so that it is inefficient for the large amount of messages. In [17], Yau et al. modified Shao et al.’s definition and extends the original PEKS/dPEKS definition 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 encrypt the message using standard symmetric encryption algorithm which is faster than public key encryption algorithm and just encrypt keyword of short size in Re-dPEKS scheme.

In the following, we describe the definition of a Bidirectional searchable proxy re-encryption with a designated tester (Re-dPEKS) [17].

A bidirectional searchable proxy re-encryption with a designated tester (Re-dPEKS) scheme consists of the following algorithms where gp denotes a set of global parameters:

1) GlobalSetup(λ): This algorithm takes a security parameter λ as input, and generate a global parameter gp.
2) KeyGenServer(gp): This algorithm takes input gp, outputs a pair of public and secret keys (pkS, skS), of server S.
3) KeyGenReceiver(gp): This algorithm takes input gp, and generates a pair of public and secret keys (pkR, skR), of receiver Rj.
4) ReKeyGen(gp, skRi, skRj): On input gp, two private keys skRi, skRj, where i ≠ j, it outputs the bidirectional re-encryption key rkRi↔Rj for receiver Rj.
5) dPEKS(gp, pkR, pkS, w): This algorithm takes as input, gp, the receiver’s public key, pkR, the server’s public key, pkS, and a selected keyword, w. It returns a dPEKS ciphertext, C of w.
6) Re-dPEKS(gp, rkRi↔Rj, C, w): On input gp, a re-encryption key rkRi↔Rj and a dPEKS ciphertext C, this algorithm returns a re-encryption dPEKS ciphertext C′, of w for receiver Rj.
7) dTrapdoor(gp, pkS, skR, w): This algorithm takes as input, gp, the server’s public key, pkS, the receiver’s secret key, skR, and a keyword, w. It then generates a trapdoor, Tw.
8) dTest(gp, C, skS, Tw): This algorithm takes as input, gp, a dPEKS ciphertext, C, the server’s secret key, skS, and a trapdoor, Tw. It outputs ‘yes’ if w = w′ and ‘no’ otherwise, where C=dPEKS(gp, pkR, pkS, w).

III. ANALYSIS OF TWO DPEKS SCHEMES AGAINST KG ATTACKS

In this section, we analyze two searchable encryption schemes with a designated tester and show that they are insecure against key guessing attacks.

A. Attack on Scheme in [11]

We review Rhee et al.’s dPEKS scheme [11] and show that the scheme is insecure against KG attacks. Rhee et al.’s dPEKS scheme works as follows:

1) GlobalSetup(λ): Given a security parameter λ, it returns a global parameter gp = (G1, G2, e, H1, H2, g, KS), where KS is a keyword space.
2) KeyGenServer(gp): It randomly chooses α ∈ R Zyp and returns skS = α and pkS = (gp, ys) = (gp, gα) as a server’s pair of secret and public keys, respectively.
3) KeyGenReceiver(gp): This algorithm randomly chooses x ∈ R Zyp, and returns skR = gα as a receiver’s pair of secret and public keys, respectively.
4) dTrapdoor(gp, pkS, skR, w): This algorithm randomly picks a rand value r′ ∈ R Zyp, and outputs Tw = [T1, T2] = [y′, H1(w)1/x ⋅ g′], where w ∈ KS.
5) dPEKS(gp, pkR, pkS, w): This algorithm randomly picks a rand value r ∈ R Zyp, and outputs C = [A, B] = [(pkR)′, H2(e(ys, H1(w)′))], where w ∈ KS.
6) dTest(gp, C, skS, Tw): This algorithm compute T = (T2)α/T1 and checks if B = H2(e(A, T)). If the equality is satisfied, then output ‘1’; otherwise, output ‘0’.

We show that the scheme in [11] is not secure against KG attacks by the server. Suppose that a server is given a ciphertext C and a trapdoor Tw for a keyword w such that dTest(gp, C, skS, Tw)=1. The server can determine which keyword is used in generating C and Tw as follows:

1) The server can get g′ from T1 using its secret key skS = α;
KG attacks as follows:

5) dPEKS: The server checks if \( e(pk_R, T_2) = e(pk_R, g^r)e(g, H_1(w')) \).

If so, the guessed keyword \( w' \) is a valid keyword. Otherwise, go to 1).

B. Attack on Scheme in [12]

Rhee et al.’s another dPEKS scheme works as follows:

1) GlobalSetup(\( \lambda \)): Given a security parameter \( \lambda \), it returns a global parameter \( gp = (G_1, G_2, c, H_1, H_2, g, h, u, t, KS) \), where \( KS \) is a keyword space and \( h, u, t \in G_1 \).

2) KeyGenServer(\( gp \)): It randomly chooses \( \alpha \in_R \mathbb{Z}_p^* \), and \( Q \in_R G_1 \), and returns \( sk_S = (\alpha, pk_S = (gp, y_{s_1}, y_{s_2}, y_{s_3}) = (gp, g^\alpha, h^{1/\alpha}) \) as a server’s pair of secret and public keys, respectively.

3) KeyGenReceiver(\( gp \)): This algorithm randomly chooses \( x \in_R \mathbb{Z}_p^* \), and returns \( sk_R = (x, pk_R = (y_{R_1}, y_{R_2}, y_{R_3}) = (g^x, h^{1/x}, t^x) \) as a receiver’s pair of secret and public keys, respectively.

4) dTrapdoor(\( gp, pk_R, pk_S, w \)): This algorithm randomly picks a rand value \( r' \in_R \mathbb{Z}_p^* \), and outputs \( T_w = H_1(w')^{1/2} \), where \( w \in KS \).

5) dPEKS(\( gp, pk_R, pk_S, w \)): This algorithm randomly picks a rand value \( r \in_R \mathbb{Z}_p^* \), and outputs \( C = [A, B] = [(y_{R_1}, y_{R_2}, y_{R_3}) = (y_{R_1}, H_1(w')^{1/2}, H(y_{S}')) \), where \( w \in KS \).

6) dTest(\( gp, C, sk_S, T_w \)): This algorithm checks if \( B = H_2(e(A, T_w)) \). If the equality is satisfied, then output ‘1’; otherwise, output ‘0’.

We show that the scheme in [12] is not secure against KG attacks as follows:

1) The server guess a keyword \( w' \) in KS and compute \( H_1(w') \);

2) The server checks if \( e(y_{R_1}, T_w) = e(g, H_1(w')) \).

If so, the guessed keyword \( w' \) is a valid keyword. Otherwise, go to 1).

IV. OUR ENHANCED DPEKS SCHEME

In our dPEKS scheme, the KeyGenReceiver method generates receiver’s public key using the information of server’s public key and we modify the structure of trapdoor. Our dPEKS scheme works as follows:

1) GlobalSetup(\( \lambda \)): Given a security parameter \( \lambda \), it returns a global parameter \( gp = (G_1, G_2, c, H_1, H_2, g, h, g, t, KS) \), where \( KH \) : \( \{0,1\}^* \rightarrow G_1 \), \( H_2 : G_2 \rightarrow \{0,1\}^* \rightarrow G_1 \).

2) KeyGenServer(\( gp \)): This algorithm randomly chooses \( \alpha \in_R \mathbb{Z}_p^* \), and returns \( sk_S = (\alpha, pk_S = (gp, y_{s_1}, y_{s_2}, y_{s_3}) = (gp, g^\alpha, h^{1/\alpha}) \) as a server's pair of secret and public keys, respectively.

3) KeyGenReceiver(\( gp \)): This algorithm randomly chooses \( x, t \in_R \mathbb{Z}_p^* \), and returns \( sk_R = (x, t, pk_R = (y_{R_1}, y_{R_2}, y_{R_3}, y_{R_4}) = (g^x, g^{tx}, g^{tx}, y_{S}')) \) as a receiver’s pair of secret and public keys, respectively.

4) dPEKS(\( gp, pk_R, pk_S, w \)): This algorithm randomly picks a rand value \( r \in_R \mathbb{Z}_p^* \), and checks if \( e(y_{R_1}, y_{R_2}) = e(y_{R_1}, y_{S'}). \) If the equality is satisfied, then outputs \( C = [A, B] = [(y_{R_1}, y_{R_2}, y_{R_3}) = (y_{R_1}, H_1(w')^{1/2}, H(y_{S}')) \).

5) dTrapdoor(\( gp, pk_R, pk_S, sk_R, w \)): This algorithm randomly picks a rand value \( r' \in_R \mathbb{Z}_p^* \), and outputs \( T_w = [T_1, T_2] = [y_{S'}, H_1(w')^{1/2}, H(y_{S}')] \).

6) dTest(\( gp, C, sk_S, T_w \)): This algorithm computes \( T = (T_1/H(T_2)^{\alpha'}) \) and checks if \( B = H_2(e(A, T)) \). If the equality is satisfied, then output ‘1’; otherwise, output ‘0’.

Correctness: When assuming the ciphertext \( C = [A, B] = [(g^{x}t^{x}, H_2(e(y_{R_1}, H_1(w')^{1/2}, H(y_{S}'))) \) is valid for the keyword \( w \) and the trapdoor \( T_w = [T_1, T_2] = [y_{S'}, H_1(w')^{1/2}, H(y_{S}')] \) for \( w' \), the correctness of the dTest algorithm is verified as

\[
T = (T_1/H(T_2)^{\alpha'}) = H_1(w')^{1/2}/H(y_{S}') = H_1(w')^{1/2}/H(y_{S}') = H_2(e(A, T)) = H_2(e(g^{x}t^{x}, H_1(w')^{1/2})) = H_2(e(g^{x}t^{x}, H_1(w')^{1/2})) = H_2(e(y_{R_1}, H_1(w'))^{1/2})
\]

If \( w \) is identical to \( w' \), the dTest algorithm outputs ‘1’.

V. ANALYSIS

Since our scheme is similar to one in [11], we can show the security in the same manner in [11]. We show that our scheme is computationally consistent and is secure against chosen keyword attack on the base that the discrete logarithm problem and the Hash Diffie-Hellman assumption are hard. We also show that our scheme is secure against keyword guessing attacks.

A. Security against chosen keyword attack

We prove the security of our scheme under discrete logarithm assumption and HDH assumption described above.

Theorem 1. Our scheme is secure against a chosen keyword attack in Game1 assuming discrete logarithm problem is intractable.

Proof. Suppose that \( A_1 \) is a malicious server with advantage \( \epsilon \) in breaking the proposed scheme. Suppose that \( A_1 \) makes \( q_T \) dTrapdoor queries. We build a simulator \( B \) that can play Game1. The simulation proceeds as follows:

We first let the challenger sets the groups \( G_1 \) and \( G_2 \) with an efficient bilinear map \( e \) and a generator \( g \) of \( G_1 \); Simulator \( B \) is given \( g, u = g^u \in G_1 \). Its goal is to compute \( \beta \in \mathbb{Z}_p^* \).

Setup: Let \( H_1 : \{0,1\}^* \rightarrow G_1, H_2 : G_2 \rightarrow \{0,1\}^* \) be two collision-resistant hash functions. \( A_1 \) generates
(sk_S, pk_S) and sends pk_S to B, where sk_S = \alpha and pk_S = (gp, y_S) = (gp, g^{\alpha}). B sends receiver’s public key pk_R = (g, g, g^\alpha). B sends pk_R to A_1.

Query phase 1: When A_1 issues a query for the dTrapdoor that corresponds to the keyword w, B responds as follows:

B randomly chooses r’ \in R Z_p^* and computes T_1’ = g^{r’} and T_2’ = H_1(w \cdot H(g^{\alpha r’})). B responds to A_1 with trapdoor T = [T_1’, T_2’] of w.

Challenge: A_1 presents \{w_0, w_1\} and gives w_0 and w_1 to B. B chooses a random b \in \{0, 1\} and l \in R Z_p^* and computes C^* = [A^*, B^*] = [u^*, H_\alpha(e(g^a, H_1(w)^y))]. Then B sends C^* to A_1.

Query phase 2: A_1 continues making dTrapdoor queries for the trapdoor T_w for any keyword w of his choice as long as w \neq w_0, w_1. B responds to A_1 as query phase 1.

Output: A_1 outputs b’ \in \{0, 1\}. If b = b’, then B can compute \beta = l.

We omit the detailed description of probability.

**Theorem 2.** Our scheme is secure against a chosen keyword attack in Game2 assuming Hash Diffie-Hellman assumption (HDH) is intractable.

Proof. Suppose that A_2 is a outsider attacker with advantage \epsilon in breaking the proposed scheme. Suppose that A_2 makes dTrapdoor queries. We build a simulator B that can solve the HDH problem. The simulation proceeds as follows:

We first let the challenger sets the groups G_1 and G_2 with an efficient bilinear map e and a generator g of G_1. Simulator B inputs a HDH instance (g, g^\alpha, H^\beta, \eta) and H : \{0, 1\}^* \rightarrow G_1 where \eta is either H(g^{\alpha e}) or a random element in G_1.

Setup: Let H_1 : \{0, 1\}^* \rightarrow G_1 be a collision-resistant hash functions. B randomly chooses l \in R Z_p^* and sets server’s public key pk_S = (gp, y_S) = (gp, g^{\alpha}). B randomly chooses the receiver’s secret key sk_R = x.r \in R Z_p^* and sets the receiver’s public key pk_R = (y_R, y_{R_0}, y_{R_2}, y_{R_4}) = (g^{x.r}, g^{x.r^2}, g^{x.r^4}, g^{x.r^6}). This implicitly defines the secret keys as sk_S = al (note that B and A_2 can not know the values of al). By the definition of the server’s and the receiver’s public keys, the equation e(y_{R_1}, y_{R_2}) = e(y_{R_4}, y_{R_6}) is satisfied.

Query phase 1: When A_2 issues a query for the dTrapdoor that corresponds to the keyword w, B responds as follows:

B randomly chooses r’ \in R Z_p^* and computes T_1’ = g^{r’} and T_2’ = H_1(w^{1/2^{s_2}} \cdot H(y_S^{r’})). B responds to A_2 with trapdoor T = [T_1’, T_2’] of w.

Challenge: A_2 present \{w_0, w_1\} and gives w_0 and w_1 to B. B chooses a random b \in \{0, 1\} and l \in R Z_p^* and computes T = [T_1, T_2] = ((g^{r’})^{1/2^{s_2}}, H_1(w^{1/2^{s_2}} \cdot \eta)), where l \in R Z_p^* and is the value selected in the Setup phase and \eta, g^\alpha are components of HDH instance. B responds to A_2 with trapdoor T = [T_1, T_2] of w_0.

Query phase 2: A_2 continues making dTrapdoor queries for the the keyword w of his choice as long as w \neq w_0, w_1. B responds to these queries as query phase 1.

Output: A_2 outputs its guess b’ \in \{0, 1\}. If b = b’, then B outputs 1 meaning \eta = H(g^{\alpha e}) , otherwise outputs 0. As T_1 = (g^{r’})^{1/2^{s_2}}, it implicitly means that r’ = e/l, so that \eta = H((g^{\alpha e})^{1/2^{s_2}}) = H(g^{\alpha e}). If \eta = H(g^{\alpha e}), A_2 must satisfy |Pr[b = b’]/1/2| > \epsilon. On the other hand, when the input instance is uniform, then \eta and T_2 are uniform and independent over G_1 in which case Pr[b = b’] = 1/2.

Therefore, we can get

\[ Pr[B(g, g^\alpha, H(g^{\alpha e})) = 1] - Pr[B(g, g^\alpha, \eta) = 1] \geq \epsilon \]

By Theorem 1 and Theorem 2, we can get that our scheme is secure against a chosen keyword attack.

**B. Security against keyword guessing attack**

Our scheme is secure against keyword guessing attacks. Suppose A is a attacker with advantage \epsilon. Assume that T_w = [T_1, T_2] is a trapdoor. To obtain a correct keyword w from the given T_w, it should be possible that A get H_1(w^{1/2^{s_2}} or H_2(w) from T_w. Since a discrete logarithm problem is hard, A cannot easily get the unknown r’ or \alpha \in Z_p if from T_1 = y_S^\alpha where \eta = y_S^{\alpha/2^{s_2}}.

Furthermore, even tough A can compute e(y_{R_1}, T_2)/e(g, T_1) = e(y_{R_4}, H_1(w^{1/2^{s_2}})

A cannot guess w such that e(y_{R_1}, T_2)/e(g, T_1) = e(y_{R_4}, H_1(w^{1/2^{s_2}})

Instead, if A guesses a keyword \omega by checking if e(y_{R_1}, T_2)/e(g, T_1) = e(y_{R_4}, H_1(w^{1/2^{s_2}})

VI. Extension

**A. Decryptable Searchable Public Key Encryption with a Designated Tester**

In the following, we give the definition of a decryptable searchable public key encryption scheme with a designated tester(dPEKSD).

A decryptable searchable public key encryption scheme with a designated tester consists of the following polynomial-time randomized algorithms where gp denotes a set of global parameters:

1) GlobalSetup(\lambda): This algorithm takes a security parameter \lambda as input, and generate a global parameter gp.

2) KeyGenServer(gp): This algorithm takes input gp, outputs a pair of public and secret keys (pk_S, sk_S), of server S.

3) KeyGenReceiver(gp): This algorithm takes input gp, and generates a pair of public and secret keys (pk_R, sk_R), of receiver R.

4) dTrapdoor(gp, pk_S, pk_R, w): This algorithm takes as input, gp, the server’s public key, pk_S, the receiver’s secret key, sk_R, and a keyword, w. It then generates a trapdoor, T_w.

5) dPEKSD(gp, pk_R, sk_S, w): This algorithm takes as input, gp, the receiver’s public key, pk_R, the server’s
public key, $pk$, and a selected keyword, $w$. It returns a dPEKS ciphertext, $C$ of $w$.

6) dTest$(gp, C, sk, T_w)$: This algorithm takes as input, $gp$, a dPEKS ciphertext, $C$, the server’s secret key, $sk$, and a trapdoor, $T_w$. It outputs ‘yes’ if $w = w'$ and ‘no’ otherwise, where $C = dPEKS(gp, pk, sk, w)$.

7) KeywordDec$(gp, C, sk)$: This algorithm takes as input, $gp$, a dPEKS ciphertext, $C$, the receiver’s secret key, $sk$. It outputs the associated keyword $w$, where $C = dPEKS(gp, pk, sk, w)$.

The security model of dPEKS is similar to that of dPEKS except that the adversary can issue KeywordDec query in phase 1 of Game 1.

We extend our dPEKS scheme to a secure decryptable searchable public key encryption scheme with a designated tester as follows:

1) GlobalSetup($\lambda$): Given a security parameter $\lambda$, it returns a global parameter $gp = (G_1, G_2, e, H_1, H_2, H_3, g)$, where $H_1 : \{0, 1\}^* \rightarrow G_1$, $H_2 : G_2 \rightarrow \{0, 1\}^*$, $H_3 : \{0, 1\}^* \rightarrow Z_p$.

2) KeyGenServer($gp$): This algorithm randomly chooses $\alpha \in Z_p$, and returns $sk = (\alpha, pk = (gp, yS) = (gp, g^\alpha))$ as a server’s pair of secret and public keys, respectively.

3) KeyGenReceiver($gp$): This algorithm randomly chooses $x, t \in Z_p$, and returns $sk_R = (x, t)$ as a receiver’s pair of secret and public keys, respectively.

4) dPEKS$(gp, pkR, pkS, w)$: This algorithm randomly picks a random value $r \in Z_p$, and checks if $e(yR_i, yS_j) = e(yR_i, yS_j)$. If the equality is satisfied, then outputs $C = [A, B, D, E]$ where $A = \langle yR_i \rangle, B = H_2(e(yR_i, H_1(w))$, $C = w \oplus H_3(g^r), D = e(g, g)^{rk_1}, h = H_3(A||B||D)||g^r)$.

5) dTrapdoor$(gp, skR, sk, w)$: This algorithm randomly picks a random value $r' \in Z_p$, and outputs $T_w = [T_1, T_2] = [g^{x^2}, H_1(w)^{1/2} \cdot H(yS_i)]$.

6) dTest$(gp, C, skS, T_w)$: This algorithm compute $T = (T_2/H(\langle T_1 \rangle))$ and checks if $B = H_2(e(A, T))$. If the equality is satisfied, then output ‘1’; otherwise, output ‘0’.

7) KeywordDec$(gp, C, skR)$: This algorithm computes $g^r$ from $A = g^{x^2} \cdot r'$ using $sk_R = (x, t)$ and outputs $w = D \oplus H_3(g^r)$ if $E = e(g, g)^h$ where $h = H_3(A||B||D)||g^r$.

B. Searchable Proxy Re-encryption with a Designated Tester

Blaze categorize two types of re-encryption schemes: bidirectional and unidirectional scheme. We extend the proposed dPEKS scheme to a bidirectional searchable proxy re-encryption with a designated tester as follows:

1) GlobalSetup($\lambda$): Given a security parameter $\lambda$, it returns a global parameter $gp = (G_1, G_2, e, H_1, H_2, H_3, g)$, where $H_1 : \{0, 1\}^* \rightarrow G_1$, $H_2 : G_2 \rightarrow \{0, 1\}^*$, $H_3 : \{0, 1\}^* \rightarrow Z_p$.

2) KeyGenServer($gp$): This algorithm randomly chooses $\alpha \in Z_p$, and returns $sk = (\alpha, pk = (gp, yS) = (gp, g^\alpha))$ as a server’s pair of secret and public keys, respectively.

3) KeyGenReceiver($gp$): This algorithm randomly chooses $x, t \in Z_p$, and returns $sk_R = (x, t)$ as a receiver’s pair of secret and public keys, respectively.

4) ReKeyGen($gp, sk, sk_R$): On input $skR_i = x_i, t_i$, and $skR_j = x_j, t_j$, output the bidirectional re-encryption key $kr_{R_i \leftrightarrow R_j} = x_i/t_i \mod p$.

5) dPEKS$(gp, pkR, pkS, w)$: This algorithm randomly picks a random value $r \in Z_p$, and checks if $e(yR_i, yR_j) = e(yR_i, yR_j)$. If the equality is satisfied, then outputs $C = [A, B] = [(yR_i)^t, H_2(e(yR_i, H_1(w))$).

6) Re-dPEKS$(gp, pkR, pkS, C_{i\leftrightarrow j})$: On input a re-encryption key $kr_{R_i \leftrightarrow R_j}$ and a dPEKS ciphertext $C_{i\leftrightarrow j} = [A, B]$, this algorithm computes $A' = A^{kr_{R_i \leftrightarrow R_j}} = g^{xt'}$, and outputs the re-encrypted ciphertext from user $R_i$ to $R_j$ as $C_{i\leftrightarrow j} = [A', B]$.

7) dTrapdoor$(gp, sk, skR, w)$: This algorithm randomly picks a random value $r' \in Z_p$, and outputs $T_w = [T_1, T_2] = [yS_i^{x^2}, H_1(w)^{1/2} \cdot H(yS_i)]$.

8) dTest$(gp, C, skS, T_w)$: This algorithm compute $T = (T_2/H(\langle T_1 \rangle))$ and checks if $B = H_2(e(A, T))$. If the equality is satisfied, then output ‘1’; otherwise, output ‘0’.

VII. CONCLUSION

The public key encryption scheme with keyword search enables one to search encrypted data without compromising the security of the original data. In this paper, we analyze two searchable public key encryption schemes with a designated tester and suggest that they are insecure against keyword guessing attack and then we construct an enhanced scheme. We prove that the scheme is secure against chosen keyword attack and keyword guessing attack. We give the definition of decryptable searchable public key encryption scheme with a designated tester and a bidirectional searchable proxy re-encryption with designated tester.
ACKNOWLEDGMENT

This project is supported by The Key Science-Technology Project of Shandong Province(Grant No. 2011GGX10124), Outstanding Young Scientists Foundation Grant of Shandong Province(Grant No. BS2009DX018).

REFERENCES


Chengyu Hu was born in Shandong, China in 1981. He has a Ph.D. in computer science and technology (2008) from Shandong University, Jinan, China. His main interest is public key cryptography including identity based cryptography, encryption, digital signature, key agreement etc.

Pengtao Liu was born in Shandong, China in 1980. She has a MS degree in computer science and technology (2006) from Shandong University, Jinan, China. Her main interest is public key encryption, digital signature etc.