

Exposure Resilient Public-key Encryption with Keyword Search against Keyword Guessing Attack

Kaito Uemura^a and Satoshi Obana^b

Hosei University, Tokyo, Japan

Keywords: Public-key Encryption with Keyword Search, Keyword Guessing Attack, Key Exposure Resiliency, Key Update.


Abstract: Public-key Encryption with Keyword Search (PEKS), proposed by Boneh et al. in 2004, is a cryptographic primitive that enables keyword search over encrypted data. The crucial issue of PEKS is that many existing PEKS do not guarantee security against Keyword Guessing Attack (KGA). Moreover, if a key exposure occurs, PEKS cannot ensure secrecy of the ciphertext entrusted to the server. In this paper, we propose two Exposure Resilient Public-Key Encryption with Keyword Search against KGA. The proposed schemes guarantee security against KGA by slightly modifying the search process of PEKS so that it cannot be executed without the server's secret key. Furthermore, the damage of key exposure is minimized by periodically updating the key. The second scheme even tolerates corruption of a server, which is realized by dividing the power of a single server into two separate servers.


1 INTRODUCTION

Since the seminal work by Boneh et al. in 2004 (Boneh et al., 2004), Public-key Encryption with Keyword Search (PEKS) is one of the hottest topics in the area of cryptography. In addition to the basic encryption function, PEKS possesses a search function over encrypted data. In the search process in PEKS, the client with the secret key issues an encrypted search query called a trapdoor to the server for the encrypted data entrusted by the client with the public key. The server verifies the encrypted data it manages and the trapdoor with test algorithm, and if the keywords match the search is successful. PEKS can ensure the security of encrypted data as long as the trapdoor is not available to the attacker. However, it cannot ensure the security of trapdoors when the adversary mounts Keyword Guessing Attack (KGA). KGA is an attack methodology specific to PEKS. The attack is effective since anyone can execute test algorithm in the model of PEKS. In addition, PEKS does not consider the exposure of the secret key held by the client. If a key exposure occurs, the attacker will also be able to break the ciphertext. Therefore, it is necessary to minimize the damage even when the secret

key is exposed.

To resolve the above mentioned problems, we propose PEKS that is secure against KGA and also secure even in when the secret key is exposed. In the proposed scheme, the server, not the client, has the secret key, and the server uses its own secret key for the test algorithm. By such design of the protocol, only the server can execute the test algorithm, which greatly contribute in preventing KGA. In addition, the proposed scheme possesses a periodic key update function to prevent key exposure. This function enables the client to use a different public key for a certain period of time so that even if the secret key is exposed during a certain period, the damage can be localized only within that period. The encrypted data is also updated in the server without decryption so that search cannot be disabled by key update. To avoid losing the searching capability due to the key update, the ciphertext is updated in the server without decryption by using proxy re-encryption (Blaze et al., 1998) (Ateniese et al., 2006). This proposed scheme was named single-server scheme. We also propose a scheme to ensure the security of the encrypted data in the server even when the server is attacked and the administrator's authority is seized, or when the server administrator attempts to illegally obtain information. This proposed scheme is an extension of single-server scheme. If we assume that the attacker is the server,

^a  <https://orcid.org/0000-0001-5707-2265>

^b  <https://orcid.org/0000-0003-4795-4779>

the encrypted data in the server cannot be protected because the attacker can use the secret key. We employ the idea of Dual-Server PEKS proposed by Chen et al. in 2015 to protect the data in malicious server (Chen et al., 2015) (Chen et al., 2016). Namely, we prepare two servers where one server (called front server) is responsible for data management, and the other (back server) is responsible for the key management. By this splitting knowledge strategy, it is impossible for the server to break the encrypted data even if the front server is illegally accessed since it does not possess the secret key. This extended scheme was named dual-server scheme. Dual-server scheme also possesses a key update functionality to minimize the damage caused by the exposure of the server key.

2 PRELIMINARIES

In this section, we describe the basic techniques used in related work and proposed schemes.

2.1 Bilinear Map (Pairing)

The bilinear map is a two-input, one-output function. Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be cyclic groups with prime order q , and e be an efficiently computable bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. In this paper, we assume a symmetric pairing, where $\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{G}$.

2.2 Public-key Encryption with Keyword Search (PEKS)

PEKS is a cryptographic protocol that makes it possible to make a search query over encrypted data. The model consists of three entities : an encryption client, a search client, a server, and consists of two phases called an entrustment phase and a search phase. In the entrustment phase, the encryption client encrypts the keywords using the public key and entrusts them to the server. In the search phase, the search client generates the trapdoor of a search keyword using its own secret key and sends it to the server. The server executes the test algorithm for all the ciphertexts it manages and the trapdoor, and outputs the results. PEKS scheme is defined by the following algorithms.

KeyGen(1^λ): Takes as input the security parameters 1^λ , outputs the public/secret key pair (pk, sk) .

PEKS(pk, w): Takes as input the public key pk and the keyword w , outputs the searchable ciphertext C_w for w .

Trapdoor(pk, sk, w'): Takes as input the public key pk , the secret key sk , and the keyword w' , outputs

the trapdoor $T_{w'}$.

Test($pk, C_w, T_{w'}$): Takes as input the public key pk , the ciphertext $C_w = \text{PEKS}(pk, w)$, and the trapdoor $T_{w'} = \text{Trapdoor}(sk, w')$, outputs 1 when $w = w'$, otherwise outputs 0.

Several variants of PEKS have been proposed so far. Chen et al. proposed Dual-Server Public-Key Encryption with Keyword Search (DS-PEKS) (Chen et al., 2015) (Chen et al., 2016). The scheme is secure against internal KGA. Matsuzaki et al. proposed Key-Updatable Public-key Encryption with Keyword Search (KU-PEKS)(Matsuzaki et al., 2017). In KU-PEKS, in addition to the ciphertext entrustment and search processes in PEKS, the secret/public keys are periodically updated and the update key is generated.

PEKS guarantees indistinguishability against chosen keyword attack (IND-CKA), a security against active attackers who can obtain trapdoors for any keywords. However, PEKS is not secure against KGA. KGA is an attack methodology used to break trapdoor keywords. In KGA, the attacker first guesses the keyword of the target trapdoor, and then encrypts the keyword with a public key. Next, the attacker executes test algorithm taking as input the ciphertext of the guessed keywords and the target trapdoor. This process can be repeated until the keywords are matched, and if they are matched, the attack is successful. KGA is unavoidable in PEKS since anyone can execute test algorithm for any ciphertext and trapdoor.

In PEKS, the secret key is held by the search client. If the secret key is exposed, the attacker can break the ciphertext through the communication channel in the same way as KGA against a trapdoor. When a key is exposed, the secret key and public key in use at that time must be revoked and a new key pair must be distributed. To make it possible to search ciphertext with the updated key, we have to re-encrypted all ciphertext.

3 SINGLE-SERVER SCHEME

Considering the above problems, we propose PEKS against KGA with key exposure resiliency. In the proposed scheme, we restrict the execution by including the secret key as the input of the test algorithm. With such modification, only the server can execute the test algorithm while the client can generate ciphertext and trapdoor with the public key. Furthermore, in the proposed scheme, key exposure resiliency is realized by periodic key update (Matsuzaki et al., 2017). The update makes it impossible for an attacker to break the ciphertext and trapdoor encrypted with the newly updated public key even if the

secret key is exposed. Thus, the damage of key exposure is minimized within the exposure period. To ensure that clients can still search for data after the key update, the server also updates the ciphertext it holds without decrypting them. Update without decryption is realized by unidirectional and multi-hop proxy re-encryption of (Wang and Cao, 2009). This proposed scheme was named single-server scheme. In the following subsections, we describe the algorithm and structure of single-server scheme in detail.

3.1 Algorithm

A single-server scheme is defined by the following six algorithms.

Setup(1^λ): Takes as input the security parameter 1^λ , generates the system parameter P .

KeyGen(P): Takes as input the system parameters P , outputs the public/secret key pair (pk, sk) .

Enc(P, pk_i, w): Takes as input P , the public key pk_i of version i , and the keyword w , outputs the ciphertext C_i for w . The index of the key indicates the version number.

ReEnc(P, sk_{i+1}, C_i): Takes as input P , the new secret key sk_{i+1} , and the ciphertext C_i encrypted with the public key of version i , outputs the ciphertext C_{i+1} encrypted with the public key of version $i + 1$. The server deletes the old key.

Trapdoor(P, pk_i, w'): Takes as input P , the public key pk_i of version i , and the keyword w' , outputs the Trapdoor T_i for w' .

Test(P, sk_i, C_i, T_i): Takes as input P , the secret key sk_i of version i , the version-matched ciphertext C_i and trapdoor T_i , outputs 1 when $w = w'$, otherwise outputs 0.

3.2 Construction

The concrete construction of the single-server scheme is described below. The encryption algorithm Enc and the trapdoor generation algorithm Trapdoor are based on the construction of the encryption algorithm in (Wang and Cao, 2009).

Setup(1^λ) $\rightarrow P$: Let \mathbb{G}, \mathbb{G}_T be a group with prime order q , g be a generator of \mathbb{G} , and h be a random elements in $\mathbb{G} \setminus \langle g \rangle$. $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ is a bilinear map. $H : \{0, 1\}^* \rightarrow \mathbb{G}_T$ is a collision-resistant hash function. The system parameter P is $(q, g, h, \mathbb{G}, \mathbb{G}_T, e, H)$.

KeyGen(P) $\rightarrow (pk_i, sk_i)$: Pick x from \mathbb{Z}_q^* randomly and generate the public/secret key pair of version i ($pk_i = g^{x_i}, sk_i = x_i$).

Enc(P, pk_i, w) $\rightarrow C_i$: For a keyword w , pick r_1 from \mathbb{Z}_q^* randomly and generate the ciphertext of w $C_i =$

$$(c_1, c_{i,2}) = (g^{r_1}, e(h, pk_i)^{r_1} \cdot H(w)).$$

ReEnc(P, sk_{i+1}, C_i) $\rightarrow C_{i+1}$: Update the ciphertext C_i and generate the ciphertext C_{i+1} . Let C_i be $(c_1, c_{i,2})$ and compute $C_{i+1} = (c_1, c_{i+1,2}) = (c_1, c_{i,2} \cdot e(h, c_{i,1})^{-x_i + x_{i+1}})$. In the re-encryption algorithm ReEnc, the old secret key x_i is removed from the ciphertext C_i and a new secret key x_{i+1} is assigned.

Trapdoor(P, pk_i, w') $\rightarrow T_i$: For a keyword w' , pick r_2 from \mathbb{Z}_q^* randomly and generate the trapdoor of w' $T_i = (t_1, t_{i,2}) = (g^{r_2}, e(h, pk_i)^{r_2} \cdot H(w')^{-1})$

Test(P, sk_i, C_i, T_i) $\rightarrow 1$ or 0 : If the keywords in the ciphertext C_i and the trapdoor T_i match in the test algorithm Test, the keyword element is removed from $C_i \cdot T_i$. Since the server can compute $C_i \cdot T_i$ from the information it holds, it can identify keyword matches by comparing them. Do the testing as follows,

$$\begin{aligned} C_i \cdot T_i &= (c_1 \cdot t_1, c_{i,2} \cdot t_{i,2}) \\ &= (g^{r_1+r_2}, e(h, g^{x_i})^{r_1+r_2} \cdot H(w)H(w')^{-1}) \\ e(h^{x_i}, c_1 \cdot t_1) &= c_{i,2} \cdot t_{i,2} \end{aligned} \tag{1}$$

If (1) holds, outputs 1, otherwise output 0.

4 DUAL-SERVER SCHEME

Though the single-server scheme is secure against external attackers who eavesdrop the communication channel, it is vulnerable to corruption of the server. This is due to the fact that a single server can execute the test algorithm by itself. To resolve the issue, we divide the power of the server into two separate servers (called "front server" and "back server") where the front server and the back server is responsible for maintaining ciphertext and the secret key, respectively. The test algorithm is executed only when the front and back servers cooperates. This extended scheme was named dual-server scheme. By separating the storage location of the ciphertext and the secret key, if the front server is attacked, the security of the ciphertext can be guaranteed since the secret key is not there. In the entrustment phase, the encryption client entrusts the ciphertext to the front server. In the search phase, the search client first sends the trapdoor to the front server. Next, the front server executes the front test for all the ciphertext and the trapdoor, and sends the internal-testing state to the back server. Finally, the back server executes the back test for the internal-testing state and return the results to the search client. We should note that idea of dividing the power of the server has been presented in (Chen et al., 2015), though, their scheme does not

support key update to minimize the damage of key exposure. Our scheme support key update executed by collaborating front and back servers. By using the approach of assigning a random element to the update key in (Matsuzaki et al., 2017), dual-server scheme guarantees the security of the ciphertext that has already been updated at the time of key exposure. In the following subsections, we describe the algorithm and structure of dual-server scheme, and finally, we provide a security proof.

4.1 Algorithm

A single-server scheme is defined by the following eight algorithms.

Setup(1^λ), **KeyGen**(P): Same as single server scheme.

Enc(P, pk_i, w): Takes as input P , the public key pk_i of version i , and the keyword w , outputs the ciphertext $C_i^{(0)} = (c_1, c_{i,2}^{(0)})$ for w with 0 updates. The index of the key indicates the version number.

ReKeyGen(P, sk_i, sk_{i+1}, c_1): Takes as input P , the secret key sk_i of version i , the secret key sk_{i+1} of version $i+1$, and the ciphertext c_1 , outputs the update key $rk_{i \rightarrow i+1}$. The server deletes the old key.

ReEnc($P, rk_{i \rightarrow i+1}, c_{i,2}^{(k)}$): Takes as input P , the update key $rk_{i \rightarrow i+1}$, and the ciphertext $c_{i,2}^{(k)}$ encrypted with the public key of version $i+k$, outputs the ciphertext $c_{i,2}^{(k+1)}$ encrypted with the public key of version $i+k+1$.

Trapdoor(P, pk_i, w'): Takes as input P , the public key pk_i of version i , and the keyword w' , outputs the Trapdoor $T_j = (t_1, t_{j,2})$ for w' .

FrontTest($P, c_{i,2}^{(k)}, t_{j,2}$): Takes as input P , the ciphertext $c_{i,2}^{(k)}$ and trapdoor $t_{j,2}$ where $k+i=j$ holds, outputs the internal testing-state C_{ITS} .

BackTest($P, sk_j, c_1, t_1, C_{ITS}, state_j$): Takes as input P , the secret key sk_j of version j , the ciphertext c_1 , the trapdoor t_1 , and the internal-testing state C_{ITS} , outputs 1 when $w = w'$, otherwise outputs 0.

4.2 Security Model

We define the security model of dual-server scheme based on the security model in (Chen et al., 2015). This security model can also be applied to single-server schemes, where the attacker is an external third party who has lower capabilities compared to the server. The security defined in this section is indistinguishability against chosen keyword attack (IND-CKA) and indistinguishability against KGA (IND-KGA), which guarantee the security of cipher-

text and trapdoor, respectively.

IND-CKA. The game between the attacker and the challenger is shown below.

Setup: The challenger executes the **KeyGen**(λ) algorithm to generate a key pair (pk_i, sk_i) and sends a public key pk_i to the attacker.

Key Exposure Query: The attacker sends a public key pk_i other than the most recent version to the challenger. The challenger returns the corresponding secret key sk_i to the attacker.

Update KeyGen Query: The attacker sends a public key pk_i other than the most recent version and the ciphertext c_1 to the challenger. The challenger executes **ReKeyGen**(P, sk_i, sk_{i+1}, c_1), and returns the update key to the attacker which updates to version $i+1$.

Update Query: The attacker sends the ciphertext $c_{i,2}^{(k)}$ and update key $rk_{i \rightarrow i+1}$ to the challenger. The challenger executes **ReEnc**($P, rk_{i \rightarrow i+1}, c_{i,2}^{(k)}$), and returns the updated ciphertext $c_{i,2}^{(k+1)}$ to the attacker.

Test Query 1: The attacker can adaptively issue the test query for any keyword and any ciphertext. The challenger returns 1 or 0 as the test result.

Challenge: The attacker sends the public key pk_i of version i and two keywords w_0, w_1 . The challenger picks $b \in \{0, 1\}$ and generates the challenge ciphertext $\text{Enc}(P, pk_i, w_b) \rightarrow C_i^*$. The challenger returns C_i^* to the attacker.

Test Query 2: The attacker can continue issue the test query for any keyword and any ciphertext except of the challenge keywords w_0, w_1 . The challenger returns 1 or 0 as the test result to the attacker.

Output: Finally, the attacker outputs its guess $b' \in \{0, 1\}$ for b and wins the game if $b = b'$.

Let the attacker in the above game be \mathcal{A} and we define the advantage as follows.

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

IND-KGA. This game is the same as IND-CKA game except that in the Challenge phase, the challenger returns a trapdoor instead of a ciphertext to the attacker, so we omit the description of it.

Let the attacker in the game be \mathcal{A} and we define the advantage as follows.

$$Adv_{\mathcal{A}}^{\text{IND-KGA}}(\lambda) = |Pr[b = b'] - 1/2|.$$

Based on the security model defined above, we define the security of dual-server scheme as follows.

Definition 1. We say that dual-server scheme if for any polynomial time attacker \mathcal{A} , we have that

$Adv_{\mathcal{A}}^{IND-CKA}(\lambda)$ and $Adv_{\mathcal{A}}^{IND-KGA}(\lambda)$ are negligible functions of the security parameter λ .

4.3 Construction

The concrete construction of the dual-server scheme is described below. When update the key, the back server selects the random elements R_i with which the key is update, which makes the front server infeasible to attack the ciphertexts generated outside the exposure period even when the front server is corrupted. The back server keeps the list $state_i$ of combinations of random elements for cancelling the random elements accumulated in the internal-testing state.

Setup(1^λ) \rightarrow $P, KeyGen(P) \rightarrow (pk_i, sk_i)$: Same as single server scheme.

Enc(P, pk_i, w) $\rightarrow C_i^{(0)}$: For a keyword w , pick r_1 from \mathbb{Z}_q^* randomly and generate the ciphertext of w
 $C_i^{(0)} = (c_1, c_{i,2}^{(0)}) = (g^{r_1}, e(h, pk_i)^{r_1} \cdot H(w))$.

ReKeyGen(P, sk_i, sk_{i+1}, c_1) $\rightarrow rk_{i \rightarrow i+1}$: Pick R_i from \mathbb{G}_T randomly and generate the update key $rk_{i \rightarrow i+1} = e(h^{-x_i+x_{i+1}}, c_{i,1}) \cdot R_i$. For $state_1 = \emptyset$ and $state_i = \{st_{i,1}, \dots, st_{i,i-1}\}$, compute $state_{i+1} = \{st_{i+1,1}, \dots, st_{i+1,i-1}, st_{i+1,i}\} = \{st_{i,1} \cdot R_i, \dots, st_{i,i-1} \cdot R_i, R_i\}$ and store in the back server. $st_{i,j} = \prod_{k=j}^{i-1} R_k$ holds.

ReEnc($P, rk_{i \rightarrow i+1}, c_{i,2}^{(k)}$) $\rightarrow c_{i,2}^{(k+1)}$: Update the ciphertext $c_{i,2}^{(k)}$ and generate the ciphertext $c_{i,2}^{(k+1)} = c_{i,2}^{(k)} \cdot rk_{i \rightarrow i+1}$.

Trapdoor(P, pk_i, w') $\rightarrow T_i$: For a keyword w' , pick r_2 from \mathbb{Z}_q^* randomly and generate the trapdoor of w'
 $T_j = (t_1, t_{j,2}) = (g^{r_2}, e(h, pk_i)^{r_2} \cdot H(w')^{-1})$

FrontTest($P, c_{i,2}^{(j-i)}, t_{j,2}$) $\rightarrow C_{ITS}$: Generate the internal testing-state $C_{ITS} = c_{i,2}^{(j-i)} \cdot t_{j,2} = e(h, g^{x_j})^{r_1+r_2} \cdot H(w)H(w')^{-1} \cdot st_{j,i}$

BackTest($P, c_{i,1}, t_{j,1}, C_{ITS}, state_j$) $\rightarrow 1$ or 0 : For $state_j = \{st_{j,1}, \dots, st_{j,j-1}\}$, do the testing as follows,

$$e(h^{x_i}, c_{i,1} \cdot t_{i,1}) \cdot st_{j,i} = C_{ITS} \quad (2)$$

If (2) holds, outputs 1, otherwise output 0.

4.4 Security of Dual-server Scheme

Theorem 1. *Dual-server scheme is indistinguishable secure against CKA assuming DBDH is intractable.*

Proof. Suppose \mathcal{A} is an attack algorithm that has advantage ϵ in breaking the proposed scheme. Suppose \mathcal{A} makes at most q_{rk} Update KeyGen Query to OR_K and at most q_{te} Test Query to OTE (we assume q_{rk} and q_{te} are positive). We construct an algorithm B

that solves the DBDH problem with probability at least $\epsilon' = \epsilon/e(q_{rk} + q_{te} + 1)$, where e is the base of the natural logarithm. Algorithm B's running time is approximately the same as \mathcal{A} 's. Hence, if the DBDH assumption holds in \mathbb{G} then ϵ' is a negligible function and consequently ϵ must be a negligible function in the security parameter.

Let g be a generator of \mathbb{G} . Algorithm B is given $g, u_1 = g^\alpha, u_2 = g^\beta, u_3 = g^\gamma, v \in \mathbb{G}^4 \times \mathbb{G}_T$. Its goal is to output 1 if $v = e(g, g)^{\alpha\beta\gamma}$, otherwise 0. Algorithm B generates the public system parameters par of the proposed scheme as follows.

1. Pick a bilinear map group $(q, g, \mathbb{G}, \mathbb{G}_T, e)$
2. Let $h = u_2$
3. Pick a collision-resistant hash function $H : \{0, 1\}^* \rightarrow \mathbb{G}_T$
4. $par = (q, g, h, \mathbb{G}, \mathbb{G}_T, e, H)$ Algorithm B simulates the challenger and interacts with attacker \mathcal{A} as follows.

Setup: Algorithm B starts by giving \mathcal{A} the parameter par .

KeyGen Query: To respond to KeyGen Query, algorithm B keeps a list of tuples $\langle i, pk_i, x_i, ci \rangle$ called the OKG list. First, the list is empty. When \mathcal{A} queries the KeyGen oracle OKG with the version number i , the algorithm B responds as follows.

1. Outputs pk_i, x_i if the query i is already in the OKG list with tuples $\langle i, pk_i, x_i, ci \rangle$
2. Otherwise, generates the random coin $coin_i \in \{0, 1\}$, where $Pr[coin_i = 0] = \delta$
3. Picks $x_i \leftarrow_R \mathbb{Z}_q^*$. If $coin_i = 0$, computes $pk_i = u_1^{x_i}$, and if $coin_i = 1$, computes $pk_i = g^{x_i}$
4. Add a tuple $\langle i, pk_i, x_i, ci \rangle$ to the OKG list and output pk_i, x_i

In either case of $coin_i$, \mathcal{A} cannot distinguish pk_i because it is equivalent for \mathcal{A} .

Key Exposure Query: Takes as input the public key pk_i , and outputs \perp if it is the latest version. Otherwise, output the corresponding x_i from the OKG list. If it is not in the list, output \perp .

Update KeyGen Query: Takes as inputs two public keys pk_i, pk_j and the ciphertext c_1 , and returns \perp if at least one of them is the latest version. Also, if there are no the public keys in the OKG list, outputs \perp . Otherwise, if $coin_i$ corresponding to the input public key is 0, outputs failure and terminates, and if it is 1, picks the corresponding x_i, x_j from the list and computes $rk_{i \rightarrow j}$ and outputs it.

Test Query 1: Takes as input the ciphertext C_i and a keyword w . If $coin_i$ corresponding to the version number of the input ciphertext is 0, outputs failure

and terminates, and if it is 1, executes Trapdoor, FrontTest and BackTest algorithms and outputs 0 or 1.

Challenge: Algorithm \mathcal{A} generates a pair of two keywords w_0 and w_1 and sends them to the algorithm B with a public key pk^* . Algorithm B generates the challenge ciphertext C^* and returns it to \mathcal{A} as follows.

1. If the public key pk^* is not in the list of O_{KG} , then returns \perp . If it is in the list, picks the corresponding secret key x^* .
2. Takes the $coin^*$ corresponding to the given public key, and if $coin^* = 1$, outputs failure and terminates, and if $coin^* = 0$, picks $b \in \{0, 1\}$ and compute the challenge ciphertext $C^* = (c_1^*, c_2^*) = (g^\gamma, H(w_b) \cdot v^{x^*})$

Test Query 2: Algorithm B works in the same way as Test Query 1. However, if the input is w_0 or w_1 , returns \perp .

Output: \mathcal{A} outputs a guess $b' \in \{0, 1\}$ that indicates whether the challenge ciphertext C^* is $\text{Enc}(P, pk^*, w_0)$ or $\text{Enc}(P, pk^*, w_1)$. Algorithm B outputs 1 if b' is the same as b picked in Challenge (i.e., $v = e(g, g)^{\alpha\beta\gamma}$). Otherwise, B outputs 0 (i.e., v is a random value).

In this simulation challenge, if $coin^* = 0$, $v = e(g, g)^{\alpha\beta\gamma}$ and $w' = w_b$ for the challenge ciphertext generated by Algorithm B, it is a correct ciphertext for \mathcal{A} , since the test algorithm holds as follows.

$$\begin{aligned} T &= (t_1, t_2) = (g^r, e(h, pk^*)^r \cdot H(w')^{-1}) \\ C_{ITS} &= c_2^* \cdot t_2 = e(h, pk^*)^{\gamma+r} \\ e(h^{-\alpha x^*}, c_1^* \cdot t_1) &= C_{ITS} \end{aligned}$$

\mathcal{A} can output the correct b' with a probability of ϵ . Therefore, if $b = b'$ holds, B can distinguish $v = e(g, g)^{\alpha\beta\gamma}$ with ϵ . On the other hand, if v is a random value, \mathcal{A} can only distinguish the correct b' that is equal to b with probability $1/2$.

Next, we consider that B can distinguish $v = e(g, g)^{\alpha\beta\gamma}$ with a probability of at least ϵ' . We analyze the probability that B does not terminate the simulation. We define the following three events.

- \mathcal{E}_1 : B does not terminate in Update KeyGen Query
- \mathcal{E}_2 : B does not terminate in Test Query
- \mathcal{E}_3 : B does not terminate in Challenge

The probability of the occurrence of \mathcal{E}_1 is $(1 - \delta)^{q_{rk}}$. The probability of the occurrence of \mathcal{E}_2 is $(1 - \delta)^{q_{te}}$. The probability of the occurrence of \mathcal{E}_3 is δ . Therefore, we can compute $(1 - \delta)^{q_{rk} + q_{te}} \delta$ as the probability that B does not terminate the simulation. Since the probability is $1/e(q_{rk} + q_{te} + 1)$ when we optimize to a maximum, B solves the DBDH problem with a probability of at least $\epsilon/e(q_{rk} + q_{te} + 1)$. \square

Theorem 2. *Dual-server scheme is indistinguishable against KGA assuming DBDH is intractable.*

Enc and Trapdoor in dual-server scheme are equivalent, and the security models of IND-KGA and IND-CKA are the same except that B returns a trapdoor instead of a ciphertext in Challenge. Therefore, the proof of Theorem 2 is the same as the proof of Theorem 1, and we omit the details of the proof.

5 CONCLUSION

In this paper, we have proposed two Exposure Resilient Public-Key Encryption with Keyword Search against KGA. Both schemes (i.e., single-server and dual-server schemes) guarantee security against KGA by slightly modifying the test algorithm so that it cannot be executed without the server's secret key. Furthermore, the damage of key exposure is minimized by periodically updating the key. The second scheme even tolerates corruption of a server, which is realized by dividing the power of a single server into front and back servers. To construct a scheme with less computational cost will be our challenging future work.

REFERENCES

- Ateniese, G., Fu, K., Green, M., and Hohenberger, S. (2006). Improved proxy re-encryption schemes with applications to secure distributed storage. In *ACM Transactions on Information and System Security, Vol. 9, No. 1*, pages pp. 1–30.
- Blaze, M., Bleumer, G., and Strauss, M. (1998). Divertible protocols and atomic proxy cryptography. In *Advances in Cryptology, EUROCRYPT'98*, pages pp. 127–144.
- Boneh, D., Crescenzo, G. D., Ostrovsky, R., and Persiano, G. (2004). Public key encryption with keyword search. In *Advances in Cryptology, EUROCRYPT 2004*, pages pp. 506–522.
- Chen, R., Mu, Y., Yang, G., Guo, F., and Wang, X. (2015). A new general framework for secure public key encryption with keyword search. In *Information Security and Privacy, Vol. 9144*, pages pp. 59–76.
- Chen, R., Mu, Y., Yang, G., Guo, F., and Wang, X. (2016). Dual-server public-key encryption with keyword search for secure cloud storage. In *IEEE Transactions on Information Forensics and Security, Vol. 11, No. 4*, pages pp. 789–798.
- Matsuzaki, N., Anada, H., and Watanabe, Y. (2017). Key-updatable public-key encryption with keyword search : The case of public-key update model. In *Computer Security Symposium 2017*.
- Wang, H. and Cao, Z. (2009). A fully secure unidirectional and multi-use proxy re-encryption scheme. Master's thesis, Department of Computer Science and Engineering, Shanghai Jiao Tong University.