Public Key Encryption with Keyword Search Revisited

Joonsang Baek, Reihaneh Safiavi-Naini,Willy Susilo
University of Wollongong
Northfields Avenue
Wollongong NSW 2522, Australia

Abstract

The public key encryption with keyword search (PEKS) scheme recently proposed by Boneh, Di Crescenzo, Ostrovsky, and Persiano enables one to search encrypted keywords without compromising the security of the original data. In this paper, we address three important issues of a PEKS scheme, "refreshing keywords", "removing secure channel", and "processing multiple keywords", which have not been considered in Boneh et. al.'s paper. We argue that care must be taken when keywords are used frequently in the PEKS scheme as this situation might contradict the security of PEKS. We then point out the inefficiency of the original PEKS scheme due to the use of the secure channel. We resolve this problem by constructing an efficient PEKS scheme that removes secure channel. Finally, we propose a PEKS scheme that encrypts multiple keywords efficiently.

Key words: Public Key Encryption with Keyword Search, Removing Secure Channel, Refreshing Keywords, Multiple Keyword Search

1 Introduction

1.1 Basic Concept

The *Public Key Encryption with Keyword Search (PEKS)* scheme proposed by Boneh et al. [6] realizes the following scenario. Suppose Alice, who is a manager of a bank, is having a holiday and away from work. She is equipped with a smart phone that can be used to check her important emails, in case there is an urgent email that requires her attention. In this scenario, Alice should be able to *select* her important emails to be read during her holiday, but *not* all of them. Due to the importance of her email, all the emails sent to her will be encrypted using her public key. This ensures that nobody else, other than Alice, will be able to retrieve the emails directed to Alice. To enable Alice to select her important emails, she must send a "*trapdoor*" to the server, so that the server can use this information to select the emails that Alice wants to read. For instance, when Alice wants to read any "urgent" emails, a trapdoor for the keyword "urgent" will be created and sent to the server. Hence, whenever someone, say Bob, would like to send an email to Alice, he needs to encrypt his email using a standard public key encryption scheme, and then appends to the resulting ciphertext one or more what we call "PEKS ciphertexts" of each keyword, to allow Alice to select the appropriate emails.

In short, PEKS provides a mechanism that allows Alice to have the email server extract emails that contain a particular keyword by providing a trapdoor corresponding to the keyword, while the email server and other parties excluding Alice *do not* learn anything else about the email.

1.2 Related Work and Our Contributions

In the literature, there are a large number of research works related to the privacy of database data. However, as noted in [6], PEKS is different from the previous solutions in the sense that the data collected by the server is from the third parties and the data is not public, which excludes the solutions of Public Information Retrieval (PIR).

There are few papers directly related to PEKS. Shortly after Boneh et al.'s work [6], Waters et al. [12] showed that the PEKS scheme based on the bilinear pairing can be applied to build encrypted and searchable audit logs.

We note that although Boneh et al.'s [6] solution for realizing PEKS is elegant, there remained some important issues regarding the use of PEKS, which were not addressed in their paper. The issues are the following.

(1) To enable the server to perform the test on a received ciphertext for a particular keyword, Alice will provide some *trapdoor* information to the server. PEKS ensures that without this trapdoor information, the server does not learn anything about the category of the email. Also, the trapdoor of a keyword does not reveal anything about the category of any other keywords. In practice the system will be used over many rounds. In its current model of PEKS, a server that has received the trapdoor for a keyword w can store the trapdoor and use it to learn all future emails with that category. In other words the current version of PEKS is a one-time system. One may assume that the server can not memorize trapdoors but this is a very restrictive assumption and not easy to implement in practice. The paper does not specify what happens if the server memorizes the *trapdoor* information related to the keyword sent by Alice, and the protection against this situation is not discussed.

(2) The scheme in [6] uses a secure (encrypted and authenticated) channel between Alice and the email server. This is certainly not suitable for some applications as building a secure channel is usually expensive.

(3) In many situations the search will be on multiple keywords that are connected through conjunctive or disjunctive logical connectives. For example an email that contains the words "urgent" *and/or* "Monday" is looked for. However, it was not discussed how one can formalize the concept of the multiple keywords search and create the PEKS ciphertexts for multiple keywords *efficiently*, and what caution must be exercised when conjunctive or disjunctive search is conducted.

In this paper, we discuss the above issues and propose *provably secure* solutions that remove secure channel and make efficient multiple keywords encryption.

2 Preliminaries

2.1 Definition of PEKS

In PEKS, three parties called "*sender*", "*receiver*", and "*server*" are involved. The sender is a party that creates and sends encrypted keywords, which we call "PEKS ciphertexts". The server is a party that receives PEKS ciphertexts and performs search upon receiving trapdoors from the receiver. The receiver is a party that creates trapdoors and sends them to the server to find the data that it wants. In what follows, we review the formal definition of PEKS given in [6].

Definition 1 (PEKS) A public key encryption with keyword search (PEKS) scheme consists of the following algorithms.

- A Receiver Key Generation Algorithm $\mathsf{KeyGen}_{\mathsf{Receiver}}(k)$: Taking a security parameter $k \in \mathbb{N}$ as input, this algorithm generates a private and public key pair (sk_R, pk_R) of the receiver. Note that pk_R includes the security parameter, descriptions of a finite keyword space and a PEKS ciphertext space.
- A PEKS Algorithm $\mathsf{PEKS}(pk_R, w)$: Taking a receiver's public key pk_R and a keyword w as input, this algorithm generates a PEKS ciphertext S which is a searchable encryption of w. We write $S = \mathsf{PEKS}(pk_R, w)$.
- A Trapdoor Generation Algorithm $\mathsf{Trapdoor}(sk_R, w)$: Taking a receiver's private key sk_R and a keyword w as input, this algorithm generates a trapdoor T_w for the keyword w.
- A Test Algorithm $\mathsf{Test}(T_w, S)$: Taking a trapdoor T_w for a keyword w and a PEKS ciphertext $S = \mathsf{PEKS}(pk, w')$, this algorithm returns a symbol "*Correct*" if $w = w'$ and "*Incorrect*" otherwise.

In [6], a security notion for PEKS schemes, "indistinguishability of PEKS against chosen keyword attack" was introduced. This notion, which we call "IND-CKA", is reviewed in Appendix A.

2.2 The Bilinear Pairing and Bilinear Diffie-Hellman Problem

We now review the bilinear pairing which plays an important role in constructing the efficient PEKS scheme proposed in [6].

Definition 2 (Bilinear Pairing [5]) The bilinear pairing $\hat{e}$ [5] is defined over two groups of the same prime-order q denoted by $\mathcal{G}_1$ and $\mathcal{G}_2$ in which the Computational Diffie-Hellman problem is intractable. We will use an additive notation to describe the operation in $\mathcal{G}_1$ while we will use a multiplicative notation for the operation in $\mathcal{G}_2$. In practice, the group $\mathcal{G}_1$ is implemented using a group of points on certain elliptic curves, each of which has a small MOV exponent [11], and the group $\mathcal{G}_2$ will be implemented using a subgroup of the multiplicative group of a finite field. The admissible bilinear map has the following properties.

1) Bilinear: $\hat{e}(aR_1, bR_2) = \hat{e}(R_1, R_2)^{ab}$, where $R_1, R_2 \in \mathcal{G}_1$ and $a, b \in \mathbb{Z}_q^*$;
2) Non-degenerate: $\hat{e}$ does not send all pairs of points in $\mathcal{G}_1 \times \mathcal{G}_1$ to the identity in $\mathcal{G}_2$. (Hence, if R is a generator of $\mathcal{G}_1$ then $\hat{e}(R, R)$ is a generator of $\mathcal{G}_2$)
3) Computable: For all $R_1, R_2 \in \mathcal{G}_1$, the map $\hat{e}(R_1, R_2)$ is efficiently computable.

We now review the definition of the Bilinear Diffie-Hellman (BDH) problem associated with the bilinear pairings [5].

Definition 3 (BDH) Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be two groups of order q, where $q > 2^k$ is a prime. Let P be a generator of $\mathcal{G}_1$. Suppose that there exists a bilinear map $\hat{e} : \mathcal{G}_1 \times \mathcal{G}_1 \to \mathcal{G}_2$. Let A be an attacker modelled as a probabilistic Turing machine, whose running time is bounded by t which is polynomial in a security parameter k. A tries to solve the following problem: *Given* (P, aP, bP, cP) *for* $a, b, c \in \mathbb{Z}_q^*$, *compute the BDH key* $\hat{e}(P, P)^{abc}$.

We define A's success by $\mathbf{Succ}^{\mathrm{BDH}}_{\mathcal{G}_1,\mathsf{A}}(k) = \Pr[\mathsf{A}(P, aP, bP, cP) = \hat{e}(P, P)^{abc}]$. The BDH problem is said to be computationally intractable if $\mathbf{Succ}^{\mathrm{BDH}}_{\mathcal{G}_1,\mathsf{A}}(k)$ is negligible in k.

3 Refreshing Keywords

In PEKS, trapdoors should be generated for *each* keyword and if this trapdoor is not released, the server will not be able to decide which PEKS ciphertext encrypts which keyword. If we removed this condition, we could simply encrypt a keyword using the receiver's public key and the receiver can send the matching private key (through secure channel) to the server. Note that the "IND-CKA" notion (reviewed in Appendix A) indeed reflects this.

However, a contradictory situation may occur when even *provably secure* PEKS is used in practice. As a natural (not artificial) example, assume that there are three keywords, say "`high priority`", "`normal`", and "`less priority`" that are *frequently* used in the system. Now, assume that whenever the receiver sends a trapdoor, the server *stores* it in its memory. Then, at some point, the server gets trapdoors for all the keywords used within the system and can decide which PEKS ciphertext encrypts which keyword *without receiving* trapdoors from the receiver. Since the storage capacity of computers are increasing rapidly these days, even if more keywords are used, say 100, the server can store all the trapdoors generated by the receiver and can conduct search by itself.

In theory, one can avoid this problem by using a keyword only once. However, this is impractical since in many situations, users want to reuse their keywords. For example, Alice may want to encrypt a keyword "`Alice`" whenever she sends a message to Bob. One may consider another alternative that when the sender encrypts a keyword with some kind of nonce, but this makes it impossible for the receiver to generate a trapdoor if the sender does not give the receiver the nonce used when the PEKS ciphertext is created. Since one of the aims of PEKS is to make keyword search possible *without* interaction between the sender and receiver, this is also not a right solution.

Nevertheless, one possible solution for the above problem is to refresh the frequently-used keywords by attaching time period information to them. For example, a keyword $w =$ `Urgent` now becomes $w' =$ `Urgent`$||$`01/07/04`, where `01/07/04` denotes "1 July 2004". Note that the more the time period is fine grained, the better security can be achieved: In the above example, once the receiver releases a trapdoor, the server can search PEKS ciphertexts that correspond to w' without receiving a trapdoor for it until the end of "day". However, if the time frame is divided by, say, a "5 hours", the server only can do so during 5 hours.

Notice that the above method essentially makes the size of a keyword space infinite and makes it useless for the server to keep trapdoors. Hence, the security notion for PEKS schemes, IND-CKA, now becomes meaningful in reality.

4 Removing Secure Channel

In the example given in Section 1.1, the connection between Alice's smart phone to the email server is through an insecure communication channel, for example via the GPRS network. However, in PEKS schemes, trapdoors cannot be sent via public channel. This is another drawback of PEKS as heavy computational and communication loads are normally required to setup secure channel such as "Secure Socket Layer (SSL)" between the server and the receiver. In this section, we propose a mechanism to remove the secure channel in a very efficient way.

4.1 Formal Model and Security Notion

The basic idea is to make the server to keep its own private and public key pair. In order to create a PEKS ciphertext, the sender uses the server's public key as well as the receiver's public key.

The receiver then can send a trapdoor to retrieve data associated with the encrypted keyword as usual, but at this time, he can send it via a *public* channel. Upon receiving the trapdoor, the server can test whether given PEKS ciphertexts match the trapdoor using its private key. In what follows, we formally define this model.

Definition 4 ($\mathsf{SCF-PEKS}$) A Secure Channel Free Public Key Encryption with Keyword Search (SCF-PEKS) scheme consists of the following algorithms.

- A Common Parameter Generation Algorithm $\mathsf{KeyGen}_{\mathsf{Param}}(k)$: Taking a security parameter $k \in \mathbb{N}$ as input, this algorithm generates a common parameter cp.
- A Server Key Generation Algorithm $\mathsf{KeyGen}_{\mathsf{Server}}(cp)$: Taking a common parameter cp as input, this algorithm generates a private and public key pair (sk_S, pk_S) of the server.
- A Receiver Key Generation Algorithm $\mathsf{KeyGen}_{\mathsf{Receiver}}(cp)$: Taking a common parameter cp as input, this algorithm generates a private and public key pair (sk_R, pk_R) of the receiver.
- A Secure Channel Free PEKS Algorithm $\mathsf{SCF-PEKS}(cp, pk_S, pk_R, w)$: Taking a common parameter cp, a server's public key pk_S, a receiver's public key pk_R and a keyword w as input, this algorithm returns a PEKS ciphertext S which is a searchable encryption of w. We write $S = \mathsf{SCF-PEKS}(cp, pk_S, pk_R, w)$.
- A Trapdoor Generation Algorithm $\mathsf{Trapdoor}(cp, sk_R, w)$: Taking a common parameter cp, a receiver's private key sk_R and a keyword w as input, this algorithm generates a trapdoor T_w for w.
- A Test Algorithm $\mathsf{Test}(cp, T_w, sk_S, S)$: Taking a common parameter cp, a trapdoor T_w for the keyword w, the server's private key sk_S, and a PEKS ciphertext $S = \mathsf{SCF-PEKS}(pk_S, pk_R, w')$, this algorithm returns a symbol "*Correct*" if $w = w'$ and "*Incorrect*" otherwise.

Note that the common parameter generation algorithm is run by a third party, e.g, a system administrator, to generate a long-term parameter that will be used within the system.

Now, we formally define a security notion for SCF-PEKS, which we call "indistinguishability of secure channel free PEKS against chosen keyword attack (IND-SCF-CKA)". Informally, IND-SCF-CKA guarantees that the server that has not obtained the trapdoors for given keywords cannot tell which PEKS ciphertext encrypts which keyword, and the outside attacker that has not obtained the server's private key cannot make any decisions about the PEKS ciphertexts even though the attacker gets *all the trapdoors* for the keywords that it holds. (That is, the attacker can see all the trapdoors including targeting ones being sent through public channel). Note that the attack model for these two types of attackers is describe as "Game 1" and "Game 2" respectively in the following definition.

Definition 5 (IND-SCF-CKA) Let A be an attacker whose running time is bounded by t which is polynomial in a security parameter k. We consider the following two games:

Game 1: A is assumed to be a server.

Phase 1-1: The common parameter generation algorithm $\mathsf{KeyGen}_{\mathsf{Param}}(k)$, the two key generation algorithms $\mathsf{KeyGen}_{\mathsf{Receiver}}(k)$ and $\mathsf{KeyGen}_{\mathsf{Server}}(k)$ are run. A common parameter cp, private and public key pairs of the receiver and the server, which we denote by (sk_R, pk_R) and (sk_S, pk_S) respectively, are then generated. cp, pk_R, sk_S, and pk_S are given to A while sk_R is kept secret from A.
Phase 1-2: A queries a number of keywords, each of which is denoted by w, to the trapdoor generation oracle $\mathsf{Trapdoor}$ and obtains a corresponding trapdoor T_w.
Phase 1-3: A outputs a target keyword pair (w_0^*, w_1^*). Upon receiving this, the PEKS oracle PEKS chooses $\beta \in \{0,1\}$ uniformly at random and creates a target PEKS ciphertext $S^* = \mathsf{SCF-PEKS}(cp, pk_S, pk_R, w_\beta^*)$ and returns it to A.
Phase 1-4: A issues a number of trapdoor extraction queries as in Phase 1-2. The restriction here is that w_0^* and w_1^* are not allowed to be queried as trapdoor extraction queries.
Phase 1-5: A outputs its guess $\beta' \in \{0,1\}$.

We define A's success in Game 1 by $\mathbf{Succ}_{\mathsf{A}}^{Game\ 1}(k) = 2\Pr[\beta' = \beta] - 1$.

Game 2: A is assumed to be an outside attacker (including the receiver).

Phase 2-1: The common parameter generation algorithm $\mathsf{KeyGen}_{\mathsf{Param}}(k)$, the two key generation algorithms $\mathsf{KeyGen}_{\mathsf{Receiver}}(k)$ and $\mathsf{KeyGen}_{\mathsf{Server}}(k)$ are run. A common parameter cp, private and public key pairs of the receiver and the server, which we denote by (sk_R, pk_R) and (sk_S, pk_S) respectively, are then generated. cp, pk_R, sk_R, and pk_S are given to A while sk_S is kept secret from A.
Phase 2-2: A queries a number of keywords, each of which is denoted by w to the trapdoor generation oracle $\mathsf{Trapdoor}$ and obtains a corresponding trapdoor T_w.
Phase 2-3: A outputs a target keyword pair (w_0^*, w_1^*). Upon receiving this, the PEKS oracle PEKS chooses $\beta \in \{0,1\}$ uniformly at random and creates a target PEKS ciphertext $S^* = \mathsf{SCF-PEKS}(cp, pk_S, pk_R, w_\beta^*)$ and returns it to A.
Phase 2-4: A issues a number of trapdoor extraction queries as in Phase 2-2. Differently from Game 1, w_0^* and w_1^* are *allowed* to be queried as trapdoor extraction queries.
Phase 2-5: A outputs its guess $\beta' \in \{0,1\}$.

We define A's success in Game 2 by $\mathbf{Succ}_{\mathsf{A}}^{Game\ 2}(k) = 2 \cdot \Pr[\beta' = \beta] - 1$.

The $\mathsf{SCF-PEKS}$ scheme is said to be IND-SCF-CKA secure if $\mathbf{Succ}_{\mathsf{PEKS,A}}^{\mathrm{IND-MK-CKA}}(k) \stackrel{\mathrm{def}}{=} \mathbf{Succ}_{\mathsf{A}}^{Game\ i}(k)$, where i is either 1 or 2, is negligible in k.

4.2 Proposed Scheme

In this section we describe our secure channel free PEKS scheme based on the aggregation technique proposed in [7]. We then provide a security proof for the scheme in the random oracle model [3].

4.2.1 Description of the Scheme

The secure channel free PEKS scheme consists of the following algorithms:

• $\mathsf{KeyGen}_{\mathsf{Param}}(k)$: Choose two groups $\mathcal{G}_1 = \langle P \rangle$ and $\mathcal{G}_2$ of the same prime order $q \geq 2^k$. Construct a bilinear pairing $\hat{e} : \mathcal{G}_1 \times \mathcal{G}_1 \rightarrow \mathcal{G}_2$. Specify hash functions $H_1 : \{0,1\}^* \rightarrow \mathcal{G}_1^*$ and $H_2 : \mathcal{G}_2 \rightarrow \{0,1\}^k$. Return $cp = (q, \mathcal{G}_1, \mathcal{G}_2, \hat{e}, P, H_1, H_2, d_{\mathcal{W}})$ as a common parameter, where $d_{\mathcal{W}}$ denotes a description of a keyword space.

• $\mathsf{KeyGen}_{\mathsf{Server}}(cp)$: Choose $x \in \mathbb{Z}_q^*$ uniformly at random and compute $X = xP$. Choose $Q \in \mathcal{G}_1^*$ uniformly at random. Return $pk_S = (cp, Q, X)$ and sk_S=(cp, x) as the server's public and private key respectively.

• $\mathsf{KeyGen}_{\mathsf{Receiver}}(cp)$: Choose $y \in \mathbb{Z}_q^*$ uniformly at random and compute $Y = yP$. Return $pk_R = (pk_S, Y)$ and sk_R=(pk_S, y) as the receiver's public and private key respectively.

• $\mathsf{SCF} - \mathsf{PEKS}(cp, pk_S, pk_R, w)$: Choose $r \in \mathbb{Z}_q^*$ and compute $S = (U, V)$ such that $(U, V) = (rP, H_2(\kappa))$, where $\kappa = (\hat{e}(Q, X)\hat{e}(H_1(w), Y))^r$. Return S as a PEKS ciphertext.

• $\mathsf{Trapdoor}(cp, sk_R, w)$: Compute $T_w = yH_1(w)$. Return T_w as a trapdoor for w.

• $\mathsf{Test}(cp, T_w, sk_S, S)$: Check if $H_2(\hat{e}(xQ + T_w, rP)) = V$. If the equation holds return "*Correct*" and "*Incorrect*" otherwise.

Compared with the original PEKS scheme proposed in [6], the above scheme needs only one more exponentiation in group $\mathcal{G}_2$ in the PEKS ciphertext generation process and one more addition in group $\mathcal{G}_1$ in the Test process as the value "$\hat{e}(Q, X)$" and "xQ" can be precomputed by the sender and the server respectively. We note that the above scheme gives a better performance than the original PEKS scheme that uses secure channel, (even though the sender should conduct one more exponentiation) as the communication between the server and the receiver need not be encrypted, which usually involves SSL-like protocol otherwise.

4.2.2 Security Analysis

We now presents the results on the security of the above scheme. The analysis of Game 1 is similar to the one given in [6], so we only provide analysis of Game 2 in the following. (The analysis of Game 1 can be found in Appendix B).

Theorem 1 *The above scheme is IND-SCF-CKA secure in the random oracle model assuming that the BDH problem is intractable.*

Proof. Suppose that the attacker B whose running time is bounded by t' is given (q, $\mathcal{G}_1$, $\mathcal{G}_2$, $\hat{e}$, P, aP, bP, cP), where $q \geq 2^k$ as an instance for the BDH problem. B's task is to compute a BDH key $\hat{e}(P, P)^{abc}$ of aP, bP, and cP using the capability of the IND-CCA attacker A which makes q_{H_1} and q_{H_2} random oracle queries and q_T trapdoor extraction queries within running time t. B simulates each phase of Game 2 of IND-SCF-CKA as follows.

Simulation of Phase 2-1. B sets $Q = bP$, $X = cP$, and chooses two random oracles H_1 and H_2, which will be specified shortly. It chooses $y \in \mathbb{Z}_q^*$ uniformly at random and compute $Y = yP$. It returns $(q, \mathcal{G}_1, \mathcal{G}_2, \hat{e}, P, H_1, H_2)$ as a common parameter, returns (Q, X) as a public key of the server. It also returns Y and y as public and private keys of the receiver respectively. The random oracles H_1 and H_2 are controlled by B as follows:

At any time A queries w_i to the random oracle H_1, B chooses $l_i \in \mathbb{Z}_q^*$ at random, computes $L_i = l_i P$ and return it as answer. If A queries κ_i to the random oracle H_2, B chooses $V_i \in \{0,1\}^k$ at random and returns it as answer. Note that B keeps query-answer lists for these random oracles.

Simulation of Phase 2-2. If A queries w_i to the trapdoor generation oracle, B computes $T_{w_i} = yH_1(w_i)$ and returns it as answer. (Note that B does know y).

Simulation of Phase 2-3. Upon receiving (w_0^*, w_1^*) from A, B chooses $R \in \{0,1\}^k$ at random and creates a target PEKS ciphertext S^* as follows.

$$S^* = (U^*, V^*) = (aP, R).$$

B then defines $R = H_2((\hat{e}(Q,X)\hat{e}(H(w_\beta^*), Y))^a)$. Note that by definition of Q, X, and Y, we have

$$(\hat{e}(Q,X)\hat{e}(H(w_\beta)^*, Y))^a = \hat{e}(bP, cP)^a \hat{e}(lP, yP)^a = \hat{e}(P,P)^{abc}\hat{e}(lP, aP)^y$$

Simulation of Phase 2-4. B answers A's queries to the random oracles and trapdoor oracle as in the simulation of Phase 2-2. (Note that B can even answer A's trapdoor extraction queries for the target keywords w_0^* and w_1^* as it knows y).

Simulation of Phase 2-5. When A outputs its guess $\beta' \in \{0,1\}$, B picks κ in the query-answer list for the random oracle H_2 and return $\kappa/\hat{e}(lP, aP)^y$ as a BDH key.

Analysis. From the specification of the simulations given above, we know that B perfectly simulates the real attack game of Game 2 of IND-SCF-CKA. Let AskW be an event that A asks either $\kappa_0^* = H_2\big((\hat{e}(Q,X)\hat{e}(H_1(w_0^*), Y))^r\big)$ or $\kappa_1^* = H_2\big((\hat{e}(Q,X)\hat{e}(H_1(w_1^*), Y))^r\big)$ to the random oracle H_2 during the real attack. Notice that if AskW does not happen, the probability that A's guess β' equals to β is at most $1/2$. Hence, applying Bayes' rule, we have

$$\begin{aligned}\Pr[\beta' = \beta] &= \Pr[\beta' = \beta | \mathsf{AskW}]\Pr[\mathsf{AskW}] + \Pr[\beta' = \beta | \neg\mathsf{AskW}]\Pr[\neg\mathsf{AskW}] \\ &\leq \Pr[\mathsf{AskW}] + \frac{1}{2}\Pr[\neg\mathsf{AskW}] = \frac{1}{2} + \frac{1}{2}\Pr[\mathsf{AskW}].\end{aligned}$$

By definition of the success probability of A in Game 2 of IND-SCF-CKA, we then get

$$\frac{1}{2} + \frac{1}{2}\mathbf{Succ}_{\mathsf{A}}^{Game\ 2}(k) \leq \frac{1}{2} + \frac{1}{2}\Pr[\mathsf{AskW}] \Longleftrightarrow \mathbf{Succ}_{\mathsf{A}}^{Game\ 2}(k) \leq \Pr[\mathsf{AskW}].$$

In the mean time, when AskW happens, B can solve the BDH problem with probability at least by picking $\kappa_\beta^* = (\hat{e}(Q,X)\hat{e}(H_1(w_0^*), Y))^a = \hat{e}(P,P)^{abc}\hat{e}(lP, aP)^y$ from the queries to the random oracle H_2 and computing $\kappa_\beta^*/\hat{e}(lP, aP)^y$. Since q_{H_2} queries are made to the random oracle H_2, we consequently have

$$\mathbf{Succ}_{\mathcal{G}_1,\mathsf{A}}^{\mathrm{BDH}}(k) \geq \frac{1}{q_{H_2}}\Pr[\mathsf{AskW}] \geq \frac{1}{q_{H_2}}\mathbf{Succ}_{\mathsf{A}}^{Game\ 2}(k).$$

Note that the running time of B is bounded by $t' = t + (q_T + q_{H_1} + 2)O(k^3)$. □

5 Handling Multiple Keywords

In practice, one may need to relate multiple keywords to one message. As Boneh et al. [6] suggested, one can achieve this by simply creating $\mathsf{E}(pk_R, M)||\mathsf{PEKS}(pk_R, w_1)||\ldots||\mathsf{PEKS}(pk_R, w_n)$, where E denotes a secure public key encryption function. Note, however, that no formalization, efficient construction, and issues related to disjunctive and conjunctive search were given in [6]. In this section, we deal with these problems.

5.1 Formal Model and Security Notion

First, we formally define a PEKS scheme with multiple keywords as follows.

Definition 6 (MPEKS) A public key encryption with keyword search scheme consists of the following algorithms.

- A Key Generation Algorithm $\mathsf{KeyGen}(k)$: This is the same as the "Key Generation" algorithm for PEKS.
- A Randomized MPEKS Algorithm $\mathsf{MPEKS}(pk, \mathbf{w})$: Taking a receiver's public key pk and a multiple keyword $\mathbf{w} = (w_1, \ldots, w_n)$ as input, this algorithm returns a MPEKS ciphertext $\mathbf{S}$ which is a searchable encryption of $\mathbf{w}$. We write $\mathbf{S} = \mathsf{MPEKS}(pk, \mathbf{w})$.
- A Trapdoor Generation Algorithm $\mathsf{Trapdoor}(sk, is, w)$: This is the same as the "Trapdoor Generation" algorithm for PEKS.
- A Test Algorithm $\mathsf{Test}(T_w, S)$: This is the same as the "Test" algorithm for PEKS.

Now, we define a security notion for MPEKS, which we call "indistinguishability of PEKS with multiple keyword search against chosen keyword attack (IND-MK-CKA)".

Definition 7 (IND-MK-CKA) Let A be an attacker whose running time is bounded by t which is polynomial in a security parameter k. We consider the following game:

Phase 1: The key generation algorithm $\mathsf{KeyGen}_{\mathsf{Receiver}}(k)$ is run. A private and public key pair (sk_R, pk_R) is then generated. pk_R is given to A while sk_R is kept secret from A.

Phase 2: A queries a number of keywords, each of which is denoted by w to the trapdoor generation oracle $\mathsf{Trapdoor}$ and obtains a corresponding trapdoor T_w.

Phase 3: A outputs a *new* target keyword-vector pair $(\mathbf{w_0^*}, \mathbf{w_1^*})$, where $\mathbf{w_0^*} = (w_{01}^*, \ldots, w_{0n}^*)$ and $\mathbf{w_1^*} = (w_{11}^*, \ldots, w_{1n}^*)$, all of which components have not been queried in Phase 2. Upon receiving this, the MPEKS oracle MPEKS chooses $\beta \in \{0, 1\}$ uniformly at random and creates a target PEKS ciphertext vector $\mathbf{S}_\beta^* = \mathsf{MPEKS}(pk, \mathbf{w}_\beta^*)$ and returns it to A.

Phase 4: A issues a number of trapdoor extraction queries as in Phase 3. The restriction here is that the target keyword-vectors $\mathbf{w_0^*}$ and $\mathbf{w_1^*}$ are not allowed to be queried as trapdoor extraction queries.

Phase 5: A outputs its guess $\beta' \in \{0, 1\}$.

We define the attacker A's success by $\mathbf{Succ}_{\mathsf{MPEKS},\mathsf{A}}^{\mathrm{IND-MK-CKA}}(k) = 2 \cdot \Pr[\beta' = \beta] - 1$. The MPEKS scheme is said to be IND-MK-CKA secure if $\mathbf{Succ}_{\mathsf{MPEKS},\mathsf{A}}^{\mathrm{IND-MK-CKA}}(k)$ is negligible in k.

5.2 Proposed Scheme

The possible construction for MPEKS given in [6] is inefficient in terms of computation and communication overhead in that the first term "rP" should be computed n times (a number of keywords) and the length of ciphertext also increases. Using the "randomness re-use" technique for multiple recipient public key encryption proposed by Kurosawa [10], we can design an efficient PEKS scheme with multiple keywords search as follows.

5.2.1 Description of the Scheme

The scheme consists of the following algorithms:

• $\mathsf{KeyGen}_{\mathsf{Receiver}}(k)$: Choose two groups $\mathcal{G}_1 = \langle P \rangle$ and $\mathcal{G}_2$ of the same prime order $q \geq 2^k$. Construct a bilinear pairing $\hat{e} : \mathcal{G}_1 \times \mathcal{G}_1 \rightarrow \mathcal{G}_2$. Choose $y \in \mathbb{Z}_q^*$ uniformly at random and compute $Y = yP$. Additionally, specify hash functions $H_1 : \{0,1\}^* \rightarrow \mathcal{G}_1^*$ and $H_2 : \mathcal{G}_2 \rightarrow \{0,1\}^k$. Return $pk_R = (q, \mathcal{G}_1, \mathcal{G}_2, \hat{e}, P, Y, H_1, H_2)$ and sk_R=$(q, \mathcal{G}_1, \mathcal{G}_2, \hat{e}, P, y, H_1, H_2)$, as receiver's public and private key respectively.

• $\mathsf{MPEKS}(pk, \mathbf{w})$ where $\mathbf{w} = (w_1, \ldots, w_n)$: Choose $r \in \mathbb{Z}_q^*$ uniformly at random and compute $\mathbf{S} = (U, V_1, \ldots, V_n)$ such that

$$U = rP, V_1 = H_2(\hat{e}(H_1(w_1), Y)^r), \ldots, V_n = H_2(\hat{e}(H_1(w_n), Y)^r)$$

Return $\mathbf{S}$ as a MPEKS ciphertext.

• $\mathsf{Trapdoor}(sk, w)$: Compute $T_w = yH_1(w)$. Return T_w as a trapdoor for a keyword w.

• $\mathsf{Test}(T_w, (U, V_i))$ for some $i \in \{1, \ldots, n\}$: Check if $H_2(\hat{e}(T_w, U)) = V_i$. If the equation holds, return "*Correct*" and "*Incorrect*" otherwise.

5.2.2 Security Analysis

Theorem 2 *The above scheme is IND-MK-CKA secure in the random oracle model assuming that the BDH problem is intractable.*

The proof is given in Appendix C.

5.3 Discussion on Disjunctive and Conjunctive Search

In some cases, it would be necessary for Alice to use some variations of the keywords, for example "urgent" *and* "from the CEO", or "urgent" *or* "invitation". We refer this type of search as *conjunctive* and *disjunctive* search, respectively.

Notice that the construction given in Section 5.2.1 naturally brings *disjunctive* keyword search: The receiver simply sends trapdoors which are combined by a special symbol "*or*", say, $[yH_1(w_{i_1})$ "*or*" $yH_1(w_{i_2})$ "*or*",... "*or*" $yH_1(w_{i_n})]$ to the server. Upon receiving this, the server checks which PEKS ciphertexts match $yH_1(w_{i_1})$ or $yH_1(w_{i_2})$ *or* ... *or* $yH_1(w_{i_n})$.

One might think that *conjunctive* keyword search can be realized in a similar way by sending $[yH_1(w_{i_1})$ "*and*" $yH_1(w_{i_2})$ "*and*",... "*and*" $yH_1(w_{i_n})]$ to the server. However, as observed by Golle, Sttadon, and Waters [9], this method has a security problem as it leaks unnecessary information to the server. As an example, assume that the server holds three entries $[\mathsf{E}(pk, M_1), U'', V_1, V_2]$, $[\mathsf{E}(pk, M_2), U', V_2, V_3]$, and $[\mathsf{E}(pk, M_3), U''', V_1, V_3]$ in its database. (Note that U's and V's are as defined in 5.2.1. In particular, V_1, V_2, and V_3 correspond to keywords w_1, w_2, and w_3 respectively). Assume that we want to find a ciphertext that corresponds to keywords w_2 "and" w_3. As mentioned earlier, if the receiver releases $[yH_1(w_2)$ "*and*" $yH_1(w_3)]$, the server can not only learn that "$\mathsf{E}(pk, M_2)$ matches w_2 and w_3" but also that "$\mathsf{E}(pk, M_1)$ matches w_2" and "$\mathsf{E}(pk, M_3)$ matches w_3".

To avoid the above problem, the sender can create a PEKS ciphertext $(rP, H_2(\hat{e}(H_1(w_2) + H_1(w_2), Y)^r))$ for keyword w_2 and w_3. However, if a number of keywords increases, this becomes impractical as there will be a large number of combination – 2^n where n is a number of keywords.

6 Concluding Remarks and Open Problems

In this paper, we discussed three issues related to PEKS and proposed provably secure PEKS schemes that remove secure channel and encrypt multiple keywords efficiently.

An interesting open problem is to design a PEKS scheme based on a primitive other than the BDH problem. As discussed in [6], the computational primitive used to construct Cock's identity-based encryption scheme turns out to be insecure. One may think that Boneh and Boyen's [4] new identity-based encryption (IBE) scheme based on the Bilinear Diffie-Hellman Inversion problem could bring a new PEKS scheme[1], but we were able to show that this is not the case: Note that in Boneh and Boyen's new IBE scheme, a message M under the public key $\mathtt{ID} \in \mathbb{Z}_q^*$ is encrypted to a ciphertext C as follows. $C = (s\mathtt{ID}P + sX, sY, \hat{e}(P,P)^s M)$, where $X = xP$ and $Y = yP$ for random $x, y, \in \mathbb{Z}_q^*$. However, the first part $s\mathtt{ID}P + sX$, does not hide information about "ID" since one can check whether $\hat{e}(sIDP + sX, Y) = \hat{e}(ID_\beta P + X, sY)$ where $\beta \in \{0,1\}$. (Note that the Decisional Diffie-Hellman problem in this group is easy.)

The server's attack by storing trapdoors seems to be inherent weakness of PEKS. Another open problem is to find a more efficient and convenient way to refresh frequently-used keywords than the one proposed in this paper.

References

[1] M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval, *Key-Privacy in Public-Key Encryption*, In Asiacrypt '01, LNCS 2248, pages 566–582, Springer-Verlag, 2001.

[2] M. Bellare, A. Boldyreva, and S. Micali, *Public-key Encryption in a Multi-User Setting: Security Proofs and Improvements*, In Eurocrypt 2000, LNCS 1807, pp. 259–274, Springer-Verlag, 2000.

[3] M. Bellare and P. Rogaway, *Random Oracles are Practical: A Paradigm for Designing Efficient Protocols*, In ACM CCCS, pages 62–73, 1993.

[4] D. Boneh and X. Boyen, *Efficient Selective-ID Secure Identity Based Encryption Without Random Oracles*, In Eurocrypt 2004, LNCS 3027, pages 223–238, Springer-Verlag, 2004.

[5] D. Boneh and M. Franklin, *Identity-Based Encryption from the Weil Pairing*, In CRYPTO 2001, LNCS 2139, pages 213–229, Springer-Verlag, 2001.

[6] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, *Public Key Encryption with Keyword Search*, In Eurocrypt 2004, LNCS 3027, pages 506–522, Springer-Verlag, 2004.

[7] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, *Aggregate and Verifiably Encrypted Signatures from Bilinear Maps*, In Eurocrypt 2001, LNCS 2656, pages 416–432, Springer-Verlag, 2003.

[8] C. Cocks, *An Identity Based Encryption Scheme Based on Quadratic Residues*, In IMA 2001, LNCS 2260, pages 360–363, Springer-Verlag, 2001.

[9] P. Golle, J. Staddon, and B. Waters, *Secure Conjunctive Search over Encrypted Data*, In ACNS 2004, LNCS 3089, pages 31–45, Springer-Verlag, 2004.

[10] K. Kurosawa, *Multi-Recepient Public-Key Encryption with Shortened Ciphertext*, In PKC 2002, LNCS 2274, pages 48–63, Springer-Verlag, 2002.

[11] A. J. Menezes, T. Okamoto, and S. A. Vanstone: *Reducing Elliptic Curve Logarithms to a Finite Field*, IEEE Tran. on Info. Theory, Vol. 31, pages 1639–1646, IEEE, 1993.

[12] B. Waters, D. Balfanz, G. Durfee, and D. Smetters, *Building an Encrypted and Searchable Audit Log* , In Network and Distributed System Security Symposium (NDSS 2004), 2004.

[1] Indeed, it was shown in [6] that constructing a PEKS scheme is a harder problem than constructing an IBE scheme. But, as stated in the same paper, investigating existing IBE schemes could be a starting point.

A Security of PEKS against Chosen Keyword Attack

Defined in the following is the security of PEKS against chosen keyword attack, which is similar to the "key privacy" notion for public key encryption defined in [1].

Definition 8 (IND-CKA) Let A be an attacker assumed to be a probabilistic Turing machine, whose running time is bounded by t which is polynomial in a security parameter k. We now consider the following game:

Phase 1: The key generation algorithm $\mathsf{KeyGen}_{\mathsf{Receiver}}(k)$ is run. A private and public key pair (sk_R, pk_R) of the receiver is then generated. pk_R is given to A while sk_R is kept secret from A.

Phase 2: A queries a number of keywords, each of which is denoted by w to the trapdoor generation oracle $\mathsf{Trapdoor}$ and obtains a corresponding trapdoor T_w.

Phase 3: A outputs a target keyword pair (w_0^*, w_1^*). Upon receiving this, the PEKS oracle PEKS chooses $\beta \in \{0,1\}$ uniformly at random and creates a target PEKS ciphertext $S^* = \mathsf{PEKS}(pk_R, pk_S, w_\beta^*)$ and returns it to A.

Phase 4: A issues a number of trapdoor extraction queries as in Phase 3. The restriction here is that w_0^* and w_1^* are not allowed to be queried as trapdoor extraction queries.

Phase 5: A outputs its guess $\beta' \in \{0,1\}$.

We define the attacker A's success by $\mathbf{Succ}^{\mathrm{IND-CKA}}_{\mathsf{PEKS},\mathsf{A}}(k) = 2 \cdot \Pr[\beta' = \beta] - 1$. The PEKS scheme is said to be IND-CKA secure if $\mathbf{Succ}^{\mathrm{IND-CKA}}_{\mathsf{PEKS},\mathsf{A}}(k)$ is negligible in k.

B Analysis of Game 1

Proof.

Suppose that the attacker B is given $(q, \mathcal{G}_1, \mathcal{G}_2, \hat{e}, P, aP, bP, cP)$, where $q \geq 2^k$, as an instance for the BDH problem. B's task is again to compute a BDH key $\hat{e}(P,P)^{abc}$ of aP, bP, and cP using the capability of the IND-SCF-CKA attacker A.

Simulation of Phase 1-1. B sets $Y = cP$. It chooses $x \in \mathbb{Z}_q^*$ uniformly at random and compute $X = xP$. It also chooses $Q \in \mathcal{G}_1^*$ uniformly at random. It returns $(q, \mathcal{G}_1, \mathcal{G}_2, \hat{e}, P, H_1, H_2)$ as a common parameter and returns (Q, X) and x as public and private keys of the server respectively. It also returns Y as a public key of the receiver. The random oracles H_1 and H_2 are controlled by B as follows.

At any time A queries w_i to the random oracle H_1, B does the following:

- Search $\mathsf{H_1List}$ for an entry $(w_i, L_i, l_i, \delta_i)$. If it exists, return L_i as answer. Otherwise, conduct the following:
  - Choose a random coin δ_i so that $\Pr[\delta_i = 0] = 1/(q_T + 1)$.
  - Choose $l_i \in \mathbb{Z}_q^*$ uniformly at random and compute $L_i = bP + l_iP$ if $\delta_i = 0$ and $L_i = l_iP$ if $\delta_i = 1$.
  - Return L_i as answer and put $(w_i, L_i, l_i, \delta_i)$ into $\mathsf{H_1List}$.

If A queries κ_i to the random oracle H_2, B searches $\mathsf{H_2List}$ for an entry (κ_i, V_i). If it exists, return V_i as answer. Otherwise, it chooses $V_i \in \{0,1\}^k$ at random and return it as answer and put (κ_i, V_i) into $\mathsf{H_1List}$.

Simulation of Phase 1-2. If A queries w_i to the trapdoor generation oracle, B does the following:

- Conduct the above procedure for simulating the random oracle H_1 to get a tuple $(w_i, L_i, l_i, \delta_i)$. If $\delta_i = 0$, output "*Abort*" and terminate. Otherwise, do the next step.
- Compute $T_{w_i} = l_i Y$ and return it as answer. Note that T_{w_i} is a correct trapdoor with respect to the receiver's public key $Y = cP$ as $T_{w_i} = l_i Y = l_i cP = cl_i P = cL_i P = cH_1(w_i)$.

Simulation of Phase 1-3. Upon receiving (w_0^*, w_1^*) from A, B does the following.

- Run the above procedure for simulating the random oracle H_1 to get tuples $(w_0^*, L_0^*, l_0^*, \delta_0^*)$ and $(w_1^*, L_1^*, l_1^*, \delta_1^*)$. If both δ_0^* and δ_1^* are equals to 1, output "*Abort*" and terminate. Otherwise, do the next step.
- Pick $\beta \in \{0, 1\}$ at random such that $\delta_\beta^* = 0$.
- Choose $R \in \{0,1\}^k$ at random and creates a target PEKS ciphertext S^* as follows.

$$S^* = (U^*, V^*) = (aP, R).$$

- Define $R = H_2((\hat{e}(Q, X)\hat{e}(H(w_\beta)^*, Y))^a)$.

Note that by definition of Q, X, and Y, we have

$$(\hat{e}(Q, X)\hat{e}(H(w_\beta)^*, Y))^a = \hat{e}(Q, xP)^a \hat{e}(bP + l_\beta^* P, cP)^a = \hat{e}(Q, aP)^x \hat{e}(P, P)^{abc} \hat{e}(aP, cP)^{l_\beta^*}.$$

Simulation of Phase 1-4. B answers A's queries to the random oracles and trapdoor generation oracle as in the simulation of Phase 1-2.

Simulation of Phase 1-5. When A outputs its guess $\beta' \in \{0, 1\}$, B picks κ in the query-answer list for the random oracle H_2 and return $\frac{\kappa}{\hat{e}(Q,aP)^x \hat{e}(aP,cP)^{l_\beta^*}}$ as a BDH key.

Derivation of the security bound is similar to the one given in [1]. So it is omitted here. □

C Proof of Theorem 2

Proof. Suppose that the attacker B is given $(q, \mathcal{G}_1, \mathcal{G}_2, \hat{e}, P, aP, bP, cP)$, where $q \geq 2^k$, as an instance for the BDH problem. B's task is again to compute a BDH key $\hat{e}(P, P)^{abc}$ of aP, bP, and cP using the capability of the IND-MK-CKA attacker A.

Simulation of Phase 1. B sets $Y = cP$ and returns $(q, \mathcal{G}_1, \mathcal{G}_2, \hat{e}, P, Y, H_1, H_2)$ as the public key of the receiver, where H_1 and H_2 are random oracles controlled by B as follows.

At any time A queries w_i to the random oracle H_1, B does the following:

- Search $\mathsf{H_1List}$ for an entry $(w_i, L_i, l_i, \delta_i)$. If it exists, return L_i as answer. Otherwise, do the following:
  - Choose a random coin δ_i so that $\Pr[\delta_i = 0] = 1/(q_T + 1)$.
  - Choose $l_i \in \mathbb{Z}_q^*$ uniformly at random and compute $L_i = bP + l_i P$ if $\delta_i = 0$ and $L_i = l_i P$ if $\delta_i = 1$.
  - Return L_i as answer and put $(w_i, L_i, l_i, \delta_i)$ into $\mathsf{H_1List}$.

If A queries κ_i to the random oracle H_2, B searches $\mathsf{H_2List}$ for an entry (κ_i, V_i). If it exists, return V_i as answer. Otherwise, it chooses $V_i \in \{0,1\}^k$ at random and return it as answer and put (κ_i, V_i) into $\mathsf{H_1List}$.

Simulation of Phase 2. If A queries w_i to the trapdoor generation oracle, B does the following:

- Run the above algorithm for simulating random oracle H_1 to get a tuple $(w_i, L_i, l_i, \delta_i)$. If $\delta_i = 0$, output "*Abort*" and terminate. Otherwise, do the following:
  - Compute $T_{w_i} = l_i Y$ and return it as answer. Note that T_{w_i} is a correct trapdoor with respect to the receiver's public key $Y = cP$ as $T_{w_i} = l_i Y = l_i cP = cl_i P = cL_i P = cH_1(w_i)$.

Simulation of Phase 3. Upon receiving a target keyword-vector pair $(\mathbf{w_0^*}, \mathbf{w_1^*})$, where $\mathbf{w_0^*} = (w_{01}^*, \ldots, w_{0n}^*)$ and $\mathbf{w_1^*} = (w_{11}^*, \ldots, w_{1n}^*)$, B does the following.

- Choose $i \in \{1, \ldots, n\}$ at random.
- Run the above algorithm for simulating the random oracle H_1 to get tuples $(w_{0i}^*, L_{0i}^*, l_{0i}^*, \delta_{0i}^*)$ and $(w_{1i}^*, L_{1i}^*, l_{1i}^*, \delta_{1i}^*)$ corresponding to (w_{0i}^*, w_{1i}^*). If both $\delta_{0i}^* = \delta_{1i}^* = 1$, output "*Abort*" and terminate. Otherwise, do the following (We know that at least one of δ_{0i}^* and δ_{1i}^* is equal to 0).:
  - Run the above algorithm for simulating the random oracle H_1 ($2n-2$ times) to get two vectors of tuples $((w_{01}^*, L_{01}^*, l_{01}^*, \delta_{01}^*), \ldots, (w_{0i-1}^*, L_{0i-1}^*, l_{0i-1}^*, \delta_{0i-1}^*), (w_{0i+1}^*, L_{0i+1}^*, l_{0i+1}^*, \delta_{0i+1}^*), \ldots, (w_{0n}^*, L_{0n}^*, l_{0n}^*, \delta_{0n}^*))$ and $((w_{11}^*, L_{11}^*, l_{11}^*, \delta_{11}^*), \ldots, (w_{1i-1}^*, L_{1i-1}^*, l_{1i-1}^*, \delta_{1i-1}^*), (w_{1i+1}^*, L_{1i+1}^*, l_{1i+1}^*, \delta_{1i+1}^*), \ldots, (w_{01}^*, L_{1n}^*, l_{1n}^*, \delta_{1n}^*))$. If both δ_{0j}^* and δ_{1j}^* are not equal to 1 for all $j = 1, \ldots, i-1, i+1, \ldots, n$, output "*Abort*" and terminate. Otherwise, do the following:
    - Pick $\beta \in \{0,1\}$ uniformly at random such that $\delta_{\beta i}^* = 0$.
    - Choose $R_i \in \{0,1\}^k$ uniformly at random and define $R_i = H_2(\hat{e}(H(w_{\beta i}^*), Y)^a)$. create a target MPEKS ciphertext $\mathbf{S}^*$ as follows.

$$\begin{aligned} \mathbf{S}^* &= (U^*, V_1^*, \ldots, V_n^*) \\ &= (aP, H_2(\hat{e}(aP, Y)^{l_{\beta 1}^*}), \ldots, H_2(\hat{e}(aP, Y)^{l_{\beta i-1}^*}), R_i, \\ &\quad H_2(\hat{e}(aP, Y)^{l_{1-\beta i+1}^*}), \ldots, H_2(\hat{e}(aP, Y)^{l_{1-\beta n}^*}). \end{aligned}$$

    - Define $R_i = H_2(\hat{e}(H(w_{\beta i}^*), Y)^a)$.

Note that by definition of Q and Y, we have

$$\hat{e}(H(w_{\beta i}^*), Y)^a = \hat{e}(bP + l_{\beta i}^* P, cP)^a = \hat{e}(bP, cP)^a \hat{e}(l_{\beta i}^* P, cP)^a = \hat{e}(P, P)^{abc} \hat{e}(aP, cP)^{l_{\beta i}^*}.$$

Note also that

$$\hat{e}(aP, Y)^{l_{\gamma j}^*} = \hat{e}(l_{\gamma j}^* P, cP)^a = \hat{e}(H_1(w_{\gamma j}^*), cP)^a$$

for $j = 1, \ldots, i-1, i+1, \ldots, n$ and $\gamma \in \{\beta, 1-\beta\}$.

Simulation of Phase 4. B answers A's trapdoor generation queries as in the simulation of Phase 2.

Simulation of Phase 5. When A outputs its guess $\beta' \in \{0,1\}$, B picks $\kappa^*_{\beta i}$ in the $\mathsf{H_2List}$ and return $\frac{\kappa^*_{\beta i}}{\hat{e}(aP,cP)^{l^*_{\beta i}}}$ as a BDH key.

Analysis. The analysis is based on the hybrid argument [2].

First, let $\mathsf{E_1}$ and $\mathsf{E_2}$ be events that B does not abort during the simulation of the trapdoor queries and the simulation of the target MPEKS ciphertext respectively. The probability that $\mathsf{E_1}$ happens is at least $\big(1-\frac{1}{q_T+1}\big)^{q_T} \geq \frac{1}{e}$. Meanwhile, the probability that $\mathsf{E_2}$ happens is at least $\big(1-\frac{1}{q_T+1}\big)^{2n-2}\big\{\big(\frac{1}{q_T+1}\big)^2+2\big(1-\frac{1}{q_T+1}\big)\big(\frac{1}{q_T+1}\big)\big\} \geq \big(\frac{q_T}{q_T+1}\big)^{2n-2}\frac{1}{q_T+1}$. Since A never issues trapdoor queries for target keyword vectors, $\mathsf{E_1}$ and $\mathsf{E_2}$ are independent. Hence, the probability that B does not abort during the entire simulation, that is $\Pr[\mathsf{E_1} \wedge \mathsf{E_2}]$, is at least $\frac{1}{e(q_T+1)}\big(\frac{q_T}{q_T+1}\big)^{2n-2}$.

Now, let Hybrid_i be an event that in the above simulation, A outputs β' such that $\beta' = \beta$, where $i \in \{1,\ldots,n\}$ is uniformly chosen at random. In other words, Hybrid_i denotes an event that A successfully guesses the keyword of the left part of a "hybrid" MPEKS ciphertext formed with i coordinates from w_β followed by $(n-i)$ coordinates from $w_{1-\beta}$.

As long as B does not abort, A's view in the above simulation is identical to its view in the real attack. Since i is uniformly chosen, we get

$$\Pr[\mathsf{AskW}] \geq \frac{1}{n}\Pr[\mathsf{E_1} \wedge \mathsf{E_2}]\sum_{i=1}^{n}\big(\Pr[\mathsf{Hybrid}_i] - \Pr[\mathsf{Hybrid}_{i-1}]\big),$$

where AskW denotes an event that A asks either $\kappa^*_{0i} = \hat{e}(H(w^*_{0i}), Y)^r$ or $\kappa^*_{1i} = \hat{e}(H(w^*_{1i}), Y)^r$ to the random oracle H_2 during the real attack.

But, when AskW happens, B can solve the BDH problem by picking picking $\kappa^*_{\beta i}$ in the $\mathsf{H_2List}$ and computing $\frac{\kappa^*_{\beta i}}{\hat{e}(aP,cP)^{l^*_{\beta i}}}$. Since q_{H_2} queries are made to the random oracle H_2, $\mathbf{Succ}^{\mathrm{BDH}}_{\mathcal{G}_1,\mathsf{A}}(k) \geq \frac{1}{q_{H_2}}\Pr[\mathsf{AskW}]$. Now that $\sum_{i=1}^{n}\big(\Pr[\mathsf{Hybrid}_i] - \Pr[\mathsf{Hybrid}_{i-1}]\big) = \Pr[\mathsf{Hybrid}_n] - \Pr[\mathsf{Hybrid}_n] = \mathbf{Succ}^{\mathrm{IND-MK-CKA}}_{\mathsf{MPEKS,A}}(k)$. Consequently, we have

$$\mathbf{Succ}^{\mathrm{BDH}}_{\mathcal{G}_1,\mathsf{B}}(k) \geq \frac{1}{en(q_T+1)}\big(\frac{q_T}{q_T+1}\big)^{2n-2}\mathbf{Succ}^{\mathrm{IND-MK-CKA}}_{\mathsf{MPEKS,A}}(k).$$

Note that the running time of B is bounded by $t' = t + (q_T + q_{H_1} + 2n)O(k^3)$. □