Search-Efficient Public-Key Authenticated Encryption with Keyword Search Made Easy

Takeshi Yoshida[1] and Keita Emura*[1,2]

[1.]Kanazawa University, Japan.
[2.]AIST, Japan.

July 16, 2026

Abstract

Public-key authenticated encryption with keyword search (PAEKS), introduced by Huang and Li (Information Sciences 2017), was proposed to provide trapdoor privacy, whereby keyword information is protected from being revealed through trapdoors. To prevent the keyword guessing attack, however, a trapdoor needs to be generated separately for each sender, and the search complexity linearly grows with the number of senders even when searching for a single keyword. To address this inefficiency, we propose a generic construction of search-efficient PAEKS. We revisit the group-oriented approach of Wang et al. (IEEE Transactions on Information Forensics and Security 2024), in which senders are organized into sender groups. Our construction is simple yet effective where all senders belonging to the same group share a single public-secret key pair, and the search complexity depends only on the number of sender groups rather than the number of individual senders. We further introduce ciphertext indistinguishability against insiders, which ensures that no keyword information is revealed from ciphertexts, even if they are generated by other members of the same sender group. We also take into account an impossibility result by Yoshida and Emura (IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, 2026), which shows that trapdoor privacy against sender-group members cannot be achieved in the scheme of Wang et al. To address this limitation, we introduce trapdoor indistinguishability against outsiders, which guarantees that no keyword information is revealed from trapdoors generated for non-group members. Our generic construction yields search-efficient group-oriented PAEKS schemes from pairings and lattices.

1 Introduction

Public-key encryption with keyword search (PEKS) [6] provides search functionality over encrypted keywords. A sender encrypts a keyword kw using a receiver's public key, while the receiver generates a trapdoor for a keyword kw'. The test algorithm takes as input a ciphertext and a trapdoor, and checks whether $kw = kw'$. Since PEKS is a public-key searchable encryption scheme, any sender can generate a ciphertext for an arbitrary keyword. Consequently, if a trapdoor is leaked, an adversary can determine whether the trapdoor corresponds to a specific keyword kw by encrypting kw and executing the test algorithm. As a result, keyword information is inevitably revealed from trapdoors through this well-known keyword guessing attack, which reveals which keyword is being searched by the receiver.

*Corresponding Author: k-emura@se.kanazawa-u.ac.jp

To prevent the keyword guessing attack, public-key authenticated encryption with keyword search (PAEKS) [21,26] has been proposed where a sender's secret key is also introduced to encrypt a keyword. However, to prevent keyword-guessing attacks, a trapdoor must be generated separately for each sender, and the search complexity grows with the number of senders even when only a single keyword is queried. To address this inefficiency, three main approaches have been proposed so far, to the best of our knowledge.

Group-Oriented Approach [31]: By defining sender groups and generating trapdoors for each sender group, ciphertexts produced by any sender within the group can be searched. The search efficiency scales with the number of sender groups.

Transformation-Oriented Approach [24, 25, 28, 32, 38]: As in standard PAEKS, both ciphertexts and trapdoors initially depend on the sender. Subsequently, by applying an appropriate transformation, they are converted into sender-independent ciphertexts and trapdoors. After the transformation, sender-independent search becomes possible in the same manner as in PEKS.

Server-Oriented Approach [11, 13, 33]: A sender server and a receiver server are defined. A sender generates a ciphertext associated with the receiver server, while a receiver generates a trapdoor associated with the sender server. The sender server removes sender-related information from the ciphertext, and the receiver server removes receiver-related information from the trapdoor, resulting in a ciphertext and a trapdoor that are associated only with the two servers. Consequently, the search complexity no longer depends on the number of senders.

From the perspective of improving search efficiency, transformation-oriented and server-oriented approaches can be considered more desirable. However, when ciphertexts or trapdoors are transformed, additional entities (such as proxies or sender/receiver servers) are required. Introducing such entities increases the complexity of the PAEKS scheme and necessitates security models in which each entity is considered potential adversary, thereby making the security proofs more involved. Although a proxy-free scheme [25] has also been proposed, the test server still performs proxy-like operations, which means that some level of trust must still be placed in the test server. Taking these considerations into account, the group-oriented approach, IBGEKS (identity-based group encryption with keyword search) [31], which defines sender groups, can be regarded as the simplest approach with the smallest deviation from the original PAEKS construction. In IBGEKS, a group primary secret key is defined for each group (α in [31]). If a ciphertext is generated by a group member, then a trapdoor generated by the corresponding group primary secret key can be used for searching.

It is worth noting that Yoshida and Emura [35] pointed out a vulnerability in IBGEKS. Their attack exploits the fact that the trapdoor is instantiated as a BLS (Boneh-Lynn-Shacham) signature [8]. By obtaining a value that is equivalent of a BLS verification key through the registration oracle used to enroll senders into a group, or through the trapdoor oracle, they demonstrate that keyword information is leaked from trapdoors. Furthermore, as a generalization, they showed that if an IBGEKS scheme satisfies correctness, then it is impossible to construct a secure IBGEKS scheme in a security model in which an adversary is allowed to access the registration oracle. This situation is analogous to that of PEKS, where any sender belonging to the same group can generate ciphertexts by the key obtained via the registration oracle, making resistance to keyword-guessing attacks inevitably impossible. This impossibility result can be avoided by appropriately defining the security model. However, even if one were to propose a patched construction to address vulnerabilities specific to IBGEKS, such schemes would remain pairing-based and therefore would not

offer post-quantum security. From this perspective, it is desirable to provide a generic construction that can be instantiated under a variety of computational assumptions. According to the situation, a natural question is:

Can we construct a generic construction of group-oriented PAEKS?

Our contribution. In this paper, we propose a generic construction of search-efficient, group-oriented PAEKS. The building blocks of our generic construction are NIKE (non-interactive key exchange) [19], PRP (pseudo-random permutation), and PEKS. Abdalla et al. [1] showed that PEKS can be generically constructed from anonymous identity-based encryption (IBE), e.g., pairing-based IBEs [7, 9, 29]. Lattice-based IBEs are basically anonymous. NIKE can be constructed from pairings [19] or lattices [20]. As a result, our generic construction yields search-efficient group-oriented PAEKS schemes from pairings and lattices.

Key Sharing: We revisit the group-oriented approach of Wang et al.'s IBGEKS [31], in which senders are organized into sender groups. Our construction is simple yet effective where all senders belonging to the same group share a single public-secret key pair, and the search complexity depends only on the number of sender groups rather than the number of individual senders.

- We assume that the key sharing is executed, e.g., by a group key agreement (GKA) protocol [3, 10]. See Section 3 for details.
- As this key-sharing approach allows the search efficiency to be improved in a simple manner, we chose to include the phrase "Made Easy" in the paper title. This title was inspired by the work of Li and Boyen [26].
- We clarify that, in the original IBGEKS [31], a group primary secret key is defined for each group, and each member, say i, possesses a distinct secret key $H(\mathsf{ID}_i)^\alpha$ derived from the group primary secret key $\alpha \in \mathbb{Z}_p$. Due to this structure, the form of a secret key is similar to that of a trapdoor, $H(kw)^\alpha$ (a BLS signature on kw), which is also derived from the group primary secret key. This structural similarity was exploited to demonstrate a vulnerability in [35]. From this perspective, we adopt a simpler approach to realize the group-oriented design in this paper via key sharing.

Insider Full CI: We note that, adopting a group-oriented approach should require that keyword information not be leaked from ciphertexts to other members belonging to the same sender group, though Wang et al. [31] did not consider this leakage. To capture this situation, we introduce full ciphertext indistinguishability against *insiders* (insider full CI), which ensures that no keyword information is revealed from ciphertexts, even if they are generated by other members of the same sender group.

Outsider TI: We also take into account the impossibility result [35], which shows that trapdoor privacy against sender-group members cannot be achieved in the scheme of Wang et al. [31] To address this limitation, we introduce trapdoor indistinguishability against *outsiders* (outsider TI), which guarantees that no keyword information is revealed from trapdoors generated for non-group members.

Technical Overview. The building blocks of our generic construction are NIKE, PRP, and PEKS. Informally, each sender and receiver holds a NIKE key pair. Here, members of the same sender group shares the same NIKE key. A shared NIKE key $\mathsf{K}_{\mathsf{NIKE}}$ established between a sender group and a receiver is then used as the key of PRP F, and each keyword kw is transformed into pseudo-random keyword pr-kw

$$\mathsf{pr}\text{-}kw \leftarrow F_{\mathsf{K}_{\mathsf{NIKE}}}(kw)$$

Then, the sender encrypts pr-kw using the encryption algorithm of the underlying PEKS scheme, and the receiver generates a trapdoor for pr-kw using the trapdoor generation algorithm of the underlying PEKS scheme, respectively. The test algorithm is the same as that of the underlying PEKS scheme.

We focus on the fact that, in PEKS, there are no sender keys in the first place, and that anyone can encrypt a keyword using the receiver's public key, while no keyword information is leaked from the ciphertexts. In other words, even if the sender secret key shared within a sender group is given to an adversary and arbitrary encryption is allowed, no keyword information is revealed from PEKS ciphertexts. Here, the PEKS adversary declares two challenge keywords. This means that, if an PAEKS adversary has the NIKE key for the (challenge) sender group, then the adversary can derive pseudo-random keywords correspond to the challenge keyword (kw_0^*, kw_1^*). Nevertheless, since these pseudo-random keywords are regarded as the challenge keywords in PEKS, the adversary cannot obtain keyword information from the challenge ciphertext due to the security of the underlying PEKS scheme. Note that when an adversary issues a trapdoor query kw that differs from the challenge keywords (kw_0^*, kw_1^*), if either $F_{\mathsf{K}_{\mathsf{NIKE}}}(kw) = F_{\mathsf{K}_{\mathsf{NIKE}}}(kw_0^*)$ or $F_{\mathsf{K}_{\mathsf{NIKE}}}(kw) = F_{\mathsf{K}_{\mathsf{NIKE}}}(kw_1^*)$ holds, then the simulation fails. To exclude the case, it seems sufficient to employ a pseudo-random function (PRF). However, since we consider an insider adversary, we cannot rely on the security properties of NIKE or PRF. To exclude this case, we exploit the fact that a permutation guarantees that distinct inputs yield distinct outputs. This is the reason why we employ a PRP, rather than PRF, in our construction. We further require that even if two distinct PRP keys, K and K', are known, it is infeasible to find distinct inputs, kw and kw', where $F_K(kw) = F_{K'}(kw')$ holds. This can be realized via a collision resistant hash function $H : \{0,1\}^{|K|+|kw|} \to \{0,1\}^{|kw|}$ and $F_K(kw) := H(K||kw)$ with the assumption that H can be seen as a random function if K is unknown.

We remark that PEKS does not protect keyword information leaked from trapdoors. Thus, an adversary of trapdoor indistinguishability may directly obtain the pseudo-random keyword of the challenge keyword kw_b^* from the challenge trapdoor (where $b \in \{0,1\}$ is the challenge bit chosen by the challenger). Due to the pseudo-randomness of PRP, the pseudo-random keyword is randomly chosen and is independent of the challenge keywords. Since we consider an outsider adversary, the adversary does not have NIKE keys and does not know the PRP key. Thus, the adversary has no advantage to distinguish b. As a limitation, our generic construction does not provide full TI security where an adversary can send the challenge keywords to the trapdoor oracle. For example, a trapdoor generation algorithm of the underlying PEKS scheme may be deterministic (e.g., a PEKS scheme constructed via the Abdalla et al. generic construction from the Boneh-Franklin IBE scheme [7]). How to simultaneously provide both insider full CI and outsider full TI is left as future work.

Similarities and Differences from Previous Generic Constructions. Our generic construction can be viewed as a hybrid of the generic constructions of PAEKS proposed by Li and Boyen [26] and by Emura [16]. Informally, in the Li-Boyen construction, a shared NIKE key $\mathsf{K}_{\mathsf{NIKE}}$ established between a sender and a receiver is used as the key for symmetric-key equality-predicate encryption (EPE). Yoshida and Emura [34] mentioned that the Li-Boyen generic construction is not CI secure

if the challenge sender's secret key is revealed due to the symmetry of NIKE (i.e,., it does not provide insider security). Precisely, a trapdoor is usually generated by a receiver using a NIKE key established by a sender's public key and the receiver's secret key. Then, the sender also can generate a trapdoor since the NIKE key can be generated by the sender's secret key and the receiver's public key. We emphasize that this is not stated as an attack and is regarded outside of the CI security model.

In contrast, in the Emura construction, smooth projective hash functions (SPHFs) are employed to establish a shared key between a sender and a receiver via SPHF-based password-authenticated key exchange (PAKE) protocols [5, 23]. The shared PAKE key is then used as the key for a PRF, which transforms each keyword kw into a pseudo-random keyword $\mathsf{pr}\text{-}kw$. The sender subsequently encrypts $\mathsf{pr}\text{-}kw$ using the encryption algorithm of the underlying PEKS scheme, while the receiver generates a trapdoor for $\mathsf{pr}\text{-}kw$ using the corresponding trapdoor generation algorithm. The test algorithm remains identical to that of the underlying PEKS scheme.[1] As a demerit, the generic construction provides a designated receiver setting where the sender key generation algorithm takes a designated receiver's public key as input, due to SPHFs.

In our construction, the SPHF-based PAKE component in the Emura generic construction is replaced with NIKE, following the approach of the Li-Boyen construction. This replacement removes the designated receiver setting in the Emura generic construction. By employing PEKS instead of EPE in the Li-Boyen generic construction, our construction provides insider full CI security. Thus, our construction effectively combines the strengths of the two generic constructions while mitigating their limitations. Further, to provide the insider security, we replace PRF into PRP.

Related Work. Zhang et al. [38] proposed multi-user keyword-field-free conjunctive searchable encryption (MU-KFFCSE) based on a transformation-oriented approach. However, it has been shown in [18] that the MU-KFFCSE scheme fails to provide TI security. Xu et al. [32] proposed a lightweight multi-user PAEKS (LM-PAEKS) scheme based on a transformation-oriented approach. A vulnerability of the LM-PAEKS scheme was reported in [17], where keyword information can be leaked from ciphertexts, as anyone can determine whether two ciphertexts correspond to the same keyword. Li et al. proposed PAEKS schemes supporting constant trapdoor generation and fast search [24, 25], also based on a transformation-oriented approach. In [37], it is reported that keyword information can be leaked from updated ciphertexts in [25] and from ciphertexts in [24]. Yang et al. [33] reported an attack on the Cheng-Meng server-aided PAEKS scheme [13] using the cloud server's secret key. A general attack has also been presented in [36].

We conjecture that these vulnerabilities stem from the increased complexity of the model compared to standard PAEKS, where additional entities are introduced, resulting in more intricate security definitions. In particular, the non-trivial nature of the associated security proofs, as evidenced by the existence of concrete attacks, suggests potential flaws or gaps in the original arguments. Motivated by this observation, we revisit the simplest group-oriented approach. Furthermore, we examine a non-trivial aspect that has been largely overlooked, namely CI security against insider adversaries.

[1] The construction methodology was originally proposed by Liu et al. [27]. Emura [16] pointed out a flaw of their security proofs and corrected the construction.

2 Preliminaries

2.1 Pseudo-random Permutation (PRP)

Let λ be a security parameter and ℓ be a polynomial. Pseudo-random permutation is defined as $\mathsf{PRP} = \{F_k : \{0,1\}^{\ell(\lambda)} \to \{0,1\}^{\ell(\lambda)}\}$. Here, we employ the security of pseudo-randomness, which guarantees the PRP is computationally indistinguishable from a truly random function.

Definition 1 (Pseudo-randomness)**.** *The advantage of $\mathcal{A}$ is defined as*

$$\mathsf{Adv}^{\mathsf{Pseudo\text{-}random}}_{\mathsf{PRP},\mathcal{A}}(\lambda) = |\Pr[\mathcal{A}^{F_k(\cdot)} = 1] - \Pr[\mathcal{A}^{R(\cdot)} = 1]|$$

where R is a set of all permutations on $\{0,1\}^{\ell(\lambda)}$. We say that the PRP provides pseudo-randomness if for all PPT adversaries, $\mathsf{Adv}^{\mathsf{Pseudo\text{-}random}}_{\mathsf{PRP},\mathcal{A}}(\lambda)$ is negligible in λ.

In our construction, each keyword kw is transformed into the pseudo-random keyword $\mathsf{pr}\text{-}kw \leftarrow F_{\mathsf{K}_{\mathsf{NIKE}}}(kw)$. We require that even if two distinct PRP keys, K and K', are known, it is infeasible to find distinct inputs, kw and kw', where $F_K(kw) = F_{K'}(kw')$ holds. This can be realized via a collision resistant hash function $H : \{0,1\}^{|K|+|kw|} \to \{0,1\}^{|kw|}$ and $F_K(kw) := H(K||kw)$ with the assumption that H can be seen as a random function if K is unknown.

2.2 Non-Interactive Key Exchange (NIKE)

Definition 2 (Syntax of NIKE)**.** $\mathsf{NIKE.SetUp}(1^\lambda)$**:** *Given a security parameter $\lambda \in \mathbb{N}$ as input, the setup algorithm outputs the common system parameter $\mathsf{pp}_{\mathsf{NIKE}}$.*

$\mathsf{NIKE.Gen}(\mathsf{NIKE.ID})$**:** *Given an identity* $\mathsf{NIKE.ID}$*, the key generation algorithm outputs a public key* $\mathsf{NIKE.pk}$ *and a secret key* $\mathsf{NIKE.sk}$*.*

$\mathsf{NIKE.SharedKey}(\mathsf{NIKE.ID}_1, \mathsf{NIKE.pk}_1, \mathsf{NIKE.ID}_2, \mathsf{NIKE.sk}_2)$**:** *Given two identities* $\mathsf{NIKE.ID}_1, \mathsf{NIKE.ID}_2$ *and corresponding key* $\mathsf{NIKE.pk}_1, \mathsf{NIKE.sk}_2$*, the key exchange algorithm outputs a shared key* $\mathsf{K}_{\mathsf{NIKE}}$ *or* $\perp$*.*

Definition 3 (Correctness)**.** *For any* $\lambda \in \mathbb{N}$, $\mathsf{pp}_{\mathsf{NIKE}} \leftarrow \mathsf{NIKE.SetUp}(1^\lambda)$, $\mathsf{NIKE.ID}_1$, $\mathsf{NIKE.ID}_2$, $(\mathsf{NIKE.pk}_1, \mathsf{NIKE.sk}_1) \leftarrow \mathsf{NIKE.Gen}(\mathsf{NIKE.ID}_1)$, $(\mathsf{NIKE.pk}_2, \mathsf{NIKE.sk}_2) \leftarrow \mathsf{NIKE.Gen}(\mathsf{NIKE.ID}_2)$, $\mathsf{K}_{\mathsf{NIKE}} \leftarrow \mathsf{NIKE.SharedKey}(\mathsf{NIKE.ID}_1, \mathsf{NIKE.pk}_1, \mathsf{NIKE.ID}_2, \mathsf{NIKE.sk}_2)$*, and* $\mathsf{K}'_{\mathsf{NIKE}} \leftarrow \mathsf{NIKE.SharedKey}(\mathsf{NIKE.ID}_2, \mathsf{NIKE.pk}_2, \mathsf{NIKE.ID}_1, \mathsf{NIKE.sk}_1)$*, the NIKE scheme is correct if* $\mathsf{K}_{\mathsf{NIKE}} = \mathsf{K}'_{\mathsf{NIKE}}$*.*

Cash, Kiltz, and Shoup [12] (CKS) proposed a formal security model for NIKE, which was subsequently extended by Freire et al. [19]. The CKS model guarantees that the shared key established by a NIKE scheme is indistinguishable from a random key. It has two variants: the passive model and the active model. In the honest key registration (HKR) setting, a passive adversary is restricted to using only honestly generated public keys, whereas in the dishonest key registration (DKR) setting, an active adversary is allowed to use arbitrarily generated (possibly malicious) public keys. Li and Boyen [26] defined both models but relied only on the HKR-CKS model in their security proof, since security in the DKR-CKS model implies security in the HKR-CKS model. For the same reason, we focus solely on the HKR-CKS model in this paper.

Definition 4 (HKR-CKS)**.** *$\mathcal{C}$ runs* $\mathsf{pp}_{\mathsf{NIKE}} \leftarrow \mathsf{NIKE.SetUp}(1^\lambda)$ *and sends* $\mathsf{pp}_{\mathsf{NIKE}}$ *to $\mathcal{A}$. $\mathcal{C}$ defines the oracle list $\mathcal{L} \leftarrow \emptyset$. The following oracle is available to $\mathcal{A}$ through this game.*

Register honest user ID oracle: $\mathcal{A}$ *sends* $\mathsf{NIKE.ID}$ *to* $\mathcal{C}$. $\mathcal{C}$ *runs* $(\mathsf{NIKE.pk}, \mathsf{NIKE.sk}) \leftarrow \mathsf{NIKE.Gen}(\mathsf{NIKE.ID})$ *and returns* $\mathsf{NIKE.pk}$. $\mathcal{C}$ *records the tuple* $\mathcal{L} \leftarrow \mathcal{L} \cup \{\mathsf{NIKE.ID}\}$.

Test oracle: $\mathcal{A}$ *sends two identities* $\mathsf{NIKE.ID}_1, \mathsf{NIKE.ID}_2$ *to* $\mathcal{C}$. *If* $\mathsf{NIKE.ID}_1 = \mathsf{NIKE.ID}_2$, $\mathcal{C}$ *returns* $\perp$, *else if* $\mathsf{NIKE.ID}_1 \notin \mathcal{L}$ *or* $\mathsf{NIKE.ID}_2 \notin \mathcal{L}$, $\mathcal{C}$ *returns* $\perp$. *Otherwise,* $\mathcal{C}$ *chooses* $b \xleftarrow{\$} \{0,1\}$. $\mathcal{C}$ *runs* $\mathsf{K}_{\mathsf{NIKE},0} \leftarrow \mathsf{NIKE.SharedKey}(\mathsf{NIKE.ID}_1, \mathsf{NIKE.pk}_1, \mathsf{ID}_2, \mathsf{NIKE.NIKE.sk}_2)$ *and generate a random key* $\mathsf{K}_{\mathsf{NIKE},1} \leftarrow \{0,1\}^\lambda$. *Finally,* $\mathcal{C}$ *returns* $\mathsf{K}_{\mathsf{NIKE},b}$ *to* $\mathcal{A}$.

Guess: $\mathcal{A}$ *outputs* $b' \in \{0,1\}$. *The advantage of* $\mathcal{A}$ *is defined as*

$$\mathsf{Adv}^{\mathsf{HKR\text{-}CKS}}_{\mathsf{NIKE},\mathcal{A}}(\lambda) = |\Pr[b = b'] - 1/2|$$

We say that the NIKE scheme is secure if for all PPT adversaries, $\mathsf{Adv}^{\mathsf{HKR\text{-}CKS}}_{\mathsf{NIKE},\mathcal{A}}(\lambda)$ *is negligible in* λ.

2.3 Public-Key Encryption with Keyword Search (PEKS)

Let $\mathsf{PEKS} = (\mathsf{PEKS.SetUp}, \mathsf{PEKS.KGen}, \mathsf{PEKS.Enc}, \mathsf{PEKS.Trapdoor}, \mathsf{PEKS.Test})$ be a PEKS scheme defined as follows.

Definition 5 (Syntax of PEKS)**.** $\mathsf{PEKS.SetUp}(1^\lambda)$**:** *Given a security parameter* $\lambda \in \mathbb{N}$ *as input, the setup algorithm outputs the common system parameter* $\mathsf{pp}_{\mathsf{PEKS}}$.

$\mathsf{PEKS.KGen}(\mathsf{pp}_{\mathsf{PEKS}})$**:** *Given* $\mathsf{pp}_{\mathsf{PEKS}}$ *as input, the key generation algorithm outputs a public key* $\mathsf{pk}_{\mathsf{PEKS}}$ *and a secret key* $\mathsf{sk}_{\mathsf{PEKS}}$.

$\mathsf{PEKS.Enc}(\mathsf{pp}_{\mathsf{PEKS}}, \mathsf{pk}_{\mathsf{PEKS}}, kw)$**:** *Given* $\mathsf{pp}_{\mathsf{PEKS}}, \mathsf{pk}_{\mathsf{PEKS}}$ *and a keyword* kw *as input, the encryption algorithm outputs a ciphertext* $\mathsf{C}_{\mathsf{PEKS},kw}$.

$\mathsf{PEKS.Trapdoor}(\mathsf{pp}_{\mathsf{PEKS}}, \mathsf{sk}_{\mathsf{PEKS}}, kw)$**:** *Given* $\mathsf{pp}_{\mathsf{PEKS}}, \mathsf{sk}_{\mathsf{PEKS}}$ *and* kw *as input, the trapdoor algorithm outputs a trapdoor* $\mathsf{T}_{\mathsf{PEKS},kw}$.

$\mathsf{PEKS.Test}(\mathsf{C}_{\mathsf{PEKS},kw}, \mathsf{T}_{\mathsf{PEKS},kw})$**:** *Given* $\mathsf{C}_{\mathsf{PEKS},kw}, \mathsf{T}_{\mathsf{PEKS},kw}$ *as input, the test algorithm outputs 1 or 0.*

Definition 6 (Correctness)**.** *It guarantees that honestly generated ciphertext can be searched (the* $\mathsf{PEKS.Test}$ *algorithm outputs 1) by an honestly generated trapdoor for the same keyword. For any* $\lambda \in \mathbb{N}, \mathsf{pp}_{\mathsf{PEKS}} \leftarrow \mathsf{SetUp}(1^\lambda), (\mathsf{pk}_{\mathsf{PEKS}}, \mathsf{sk}_{\mathsf{PEKS}}) \leftarrow \mathsf{PEKS.KGen}(\mathsf{pp}_{\mathsf{PEKS}})$ *and* $kw \in \mathcal{KW}$, *the PEKS scheme is correct if* $1 \leftarrow \mathsf{Test}(\mathsf{PEKS.Enc}(\mathsf{pp}_{\mathsf{PEKS}}, \mathsf{pk}_{\mathsf{PEKS}}, kw), \mathsf{PEKS.Trapdoor}(\mathsf{pp}_{\mathsf{PEKS}}, \mathsf{sk}_{\mathsf{PEKS}}, kw))$.

Definition 7 (Consistency)**.** *It guarantees that the* $\mathsf{PEKS}.\mathit{Test}$ *algorithm outputs 0 if the keyword specified during encryption differs from the keyword specified during trapdoor generation.* $\mathcal{C}$ *runs* $\mathsf{pp}_{\mathsf{PEKS}} \leftarrow \mathsf{PEKS.SetUp}(1^\lambda)$, $(\mathsf{pk}_{\mathsf{PEKS}}, \mathsf{sk}_{\mathsf{PEKS}}) \leftarrow \mathsf{PEKS.KGen}(\mathsf{pp}_{\mathsf{PEKS}})$ *and sends* $\mathsf{pk}_{\mathsf{PEKS}}$ *to* $\mathcal{A}$. *Then,* $\mathcal{A}$ *outputs* $(kw, kw') \in \mathcal{KW}^2$. $\mathcal{A}$ *wins if* $1 \leftarrow \mathsf{PEKS.Test}(\mathsf{C}_{\mathsf{PEKS},kw}, \mathsf{T}_{\mathsf{PEKS},kw'})$ *and* $kw \neq kw'$ *hold where* $\mathsf{C}_{\mathsf{PEKS},kw} \leftarrow \mathsf{PEKS.Enc}(\mathsf{pp}_{\mathsf{PEKS}}, \mathsf{pk}_{\mathsf{PEKS}}, kw)$ *and* $\mathsf{T}_{\mathsf{PEKS},kw'} \leftarrow \mathsf{PEKS.Trapdoor}(\mathsf{pp}_{\mathsf{PEKS}}, \mathsf{sk}_{\mathsf{PEKS}}, kw')$. *The advantage of* $\mathcal{A}$ *is defined as*

$$\mathsf{Adv}^{\mathsf{consist}}_{\mathsf{PEKS},\mathcal{A}}(\lambda) = \Pr[\mathcal{A}\ \mathit{wins}]$$

We say that the PEKS scheme is consistent if for all PPT adversaries, $\mathsf{Adv}^{\mathsf{consist}}_{\mathsf{PEKS},\mathcal{A}}(\lambda)$ *is negligible in* λ.

Definition 8 (IND-CKA). *$\mathcal{C}$ runs* $\mathsf{pp}_{\mathsf{PEKS}} \leftarrow \mathsf{PEKS.SetUp}(1^\lambda), (\mathsf{pk}_{\mathsf{PEKS}}, \mathsf{sk}_{\mathsf{PEKS}}) \leftarrow \mathsf{PEKS.KGen}(\mathsf{pp}_{\mathsf{PEKS}})$ *and sends* $\mathsf{pp}_{\mathsf{PEKS}}, \mathsf{pk}_{\mathsf{PEKS}}$ *to $\mathcal{A}$. The trapdoor oracle is available to $\mathcal{A}$ as follows.*

Phase I: *The following oracle is available to $\mathcal{A}$.*

> **Trapdoor oracle:** *$\mathcal{A}$ sends kw to $\mathcal{C}$. $\mathcal{C}$ runs* $\mathsf{T}_{\mathsf{PEKS},kw} \leftarrow \mathsf{PEKS.Trapdoor}(\mathsf{pp}_{\mathsf{PEKS}}, \mathsf{sk}_{\mathsf{PEKS}}, kw)$ *and returns* $\mathsf{T}_{\mathsf{PEKS},kw}$ *to $\mathcal{A}$.*

Challenge Phase: *$\mathcal{A}$ sends $\mathcal{C}$ two challenge keywords (kw_0^*, kw_1^*) which were not sent to the trapdoor oracle. $\mathcal{C}$ chooses $b \xleftarrow{\$} \{0,1\}$ and returns the challenge ciphertext* $\mathsf{C}_{\mathsf{PEKS},kw_b^*} \leftarrow \mathsf{PEKS.Enc}(\mathsf{pp}_{\mathsf{PEKS}}, \mathsf{pk}_{\mathsf{PEKS}}, kw_b^*)$.

Phase II: *The following oracle is available to $\mathcal{A}$.*

> **Trapdoor oracle:** *$\mathcal{A}$ sends $kw \notin \{kw_0^*, kw_1^*\}$ to $\mathcal{C}$. $\mathcal{C}$ runs* $\mathsf{T}_{\mathsf{PEKS},kw} \leftarrow \mathsf{PEKS.Trapdoor}(\mathsf{pp}_{\mathsf{PEKS}}, \mathsf{sk}_{\mathsf{PEKS}}, kw)$ *and returns* $\mathsf{T}_{\mathsf{PEKS},kw}$ *to $\mathcal{A}$.*

Guess: *$\mathcal{A}$ outputs $b' \in \{0,1\}$. The advantage of $\mathcal{A}$ is defined as*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CKA}}_{\mathsf{PEKS},\mathcal{A}}(\lambda) = |\Pr[b = b'] - 1/2|$$

> *We say that the PEKS scheme is IND-CKA secure if for all PPT adversaries,* $\mathsf{Adv}^{\mathsf{IND\text{-}CKA}}_{\mathsf{PEKS},\mathcal{A}}(\lambda)$ *is negligible in λ.*

3 Definitions of (Group-Oriented) PAEKS

In this section, we define PAEKS. A sender's key in PAEKS is typically associated with a single sender. In contrast, we assume that a sender's key is shared among all members belonging to the same sender group.

As the simplest approach, a group representative runs the sender group key generation algorithm, $\mathsf{KGen}_{\mathsf{SG}}$, and distributes the resulting public and secret key pair $(\mathsf{pk}_{\mathsf{SG}}, \mathsf{sk}_{\mathsf{SG}})$ to the sender group members. Alternatively, employing an identity-based setting is also effective where a key generation center (KGC) issues $(\mathsf{pk}_{\mathsf{SG}}, \mathsf{sk}_{\mathsf{SG}})$ to the sender group members via the registration algorithm as in IBGEKS [31]. To remove the need for a trusted setup, the group can instead employ a GKA protocol, such as a discrete-logarithm based GKA protocol [10] or a lattice-based GKA protocol [3]. Alternatively, secure group messaging protocols, which have recently emerged as an actively studied research area (to name a few, [2, 4, 14, 15, 22, 30]), could also be employed, although this may be an overkill for our purpose. After a group key, say k, has been shared among the sender group members, each sender runs the $\mathsf{KGen}_{\mathsf{SG}}$ algorithm, treating k as the randomness where $(\mathsf{pk}_{\mathsf{SG}}, \mathsf{sk}_{\mathsf{SG}}) \leftarrow \mathsf{KGen}_{\mathsf{SG}}(\mathsf{pp}, \mathsf{ID}_{\mathsf{SG}}; k)$ and all group members share $(\mathsf{pk}_{\mathsf{SG}}, \mathsf{sk}_{\mathsf{SG}})$. Of course, in our construction it is also possible for each sender to perform key generation individually, that is, each sender group consists of a single sender. In this case, our construction operates as a standard PAEKS scheme. Hereafter, we do not explicitly describe the key-sharing procedure among sender group members.

Let $\mathsf{PAEKS} = (\mathsf{SetUp}, \mathsf{KGen}_{\mathsf{SG}}, \mathsf{KGen}_{\mathsf{R}}, \mathsf{Enc}, \mathsf{Trapdoor}, \mathsf{Test})$ be a PAEKS scheme defined as follows.

Definition 9 (Syntax of PAEKS). $\mathsf{SetUp}(1^\lambda)$**:** *Given a security parameter $\lambda \in \mathbb{N}$ as input, the setup algorithm outputs the common system parameter* pp.

$\mathsf{KGen}_{\mathsf{SG}}(\mathsf{pp}, \mathsf{ID}_{\mathsf{SG}})$**:** *Given* pp *and an identity of sender group* $\mathsf{ID}_{\mathsf{SG}}$ *as input, the sender group key generation algorithm outputs a sender group's public key* $\mathsf{pk}_{\mathsf{SG}}$ *and a sender group's secret key* $\mathsf{sk}_{\mathsf{SG}}$*. Here,* $\mathsf{ID}_{\mathsf{SG}}$ *is embedded within* $\mathsf{pk}_{\mathsf{SG}}$*, such that* $\mathsf{ID}_{\mathsf{SG}}$ *can be identified through* $\mathsf{pk}_{\mathsf{SG}}$

$\mathsf{KGen}_{\mathsf{R}}(\mathsf{pp}, \mathsf{ID}_{\mathsf{R}})$**:** *Given* pp *and a receiver's identity as input, the receiver key generation algorithm outputs a receiver's public key* pk_{R} *and a receiver's secret key* sk_{R}*. Here,* ID_{R} *is embedded within* pk_{R}*, such that* ID_{R} *can be identified through* pk_{R}

$\mathsf{Enc}(\mathsf{pp}, \mathsf{pk}_{\mathsf{SG}}, \mathsf{pk}_{\mathsf{R}}, \mathsf{sk}_{\mathsf{SG}}, kw)$**:** *Given* $\mathsf{pp}, \mathsf{pk}_{\mathsf{R}}, \mathsf{pk}_{\mathsf{SG}}, \mathsf{sk}_{\mathsf{SG}}$ *and a keyword* kw *as input, the encryption algorithm outputs a ciphertext* C_{kw}*.*

$\mathsf{Trapdoor}(\mathsf{pp}, \mathsf{pk}_{\mathsf{R}}, \mathsf{pk}_{\mathsf{SG}}, \mathsf{sk}_{\mathsf{R}}, kw)$**:** *Given* $\mathsf{pp}, \mathsf{pk}_{\mathsf{R}}, \mathsf{pk}_{\mathsf{SG}}, \mathsf{sk}_{\mathsf{R}}$ *and* kw *as input, the trapdoor generation algorithm outputs a trapdoor* T_{kw}*.*

$\mathsf{Test}(\mathsf{C}_{kw}, \mathsf{T}_{kw})$**:** *Given* $\mathsf{C}_{kw}, \mathsf{T}_{kw}$ *as input, the test algorithm outputs 1 or 0.*

Definition 10 (Correctness)**.** *For any* $\lambda \in \mathbb{N}$, $\mathsf{pp} \leftarrow \mathsf{SetUp}(1^\lambda)$, $\mathsf{ID}_{\mathsf{SG}}, \mathsf{ID}_{\mathsf{R}}$, $(\mathsf{pk}_{\mathsf{SG}}, \mathsf{sk}_{\mathsf{SG}}) \leftarrow \mathsf{KGen}_{\mathsf{SG}}(\mathsf{pp}, \mathsf{ID}_{\mathsf{SG}})$, $(\mathsf{pk}_{\mathsf{R}}, \mathsf{sk}_{\mathsf{R}}) \leftarrow \mathsf{KGen}_{\mathsf{R}}(\mathsf{pp}, \mathsf{ID}_{\mathsf{R}})$*, and* $kw \in \mathcal{KW}$*, the PAEKS scheme is correct if* $1 \leftarrow \mathsf{Test}(\mathsf{Enc}(\mathsf{pp}, \mathsf{pk}_{\mathsf{R}}, \mathsf{pk}_{\mathsf{SG}}, \mathsf{sk}_{\mathsf{SG}}, kw), \mathsf{Trapdoor}(\mathsf{pp}, \mathsf{pk}_{\mathsf{R}}, \mathsf{pk}_{\mathsf{SG}}, \mathsf{sk}_{\mathsf{R}}, kw))$

Definition 11 (Consistency)**.** *It guarantees that the* Test *algorithm outputs 0 if any of the sender, the receiver, or the keyword specified during encryption differs from the corresponding sender, receiver, or keyword specified during trapdoor generation.* $\mathcal{C}$ *runs* $\mathsf{pp} \leftarrow \mathsf{SetUp}(1^\lambda)$, $(\mathsf{pk}_{\mathsf{SG}_0}, \mathsf{sk}_{\mathsf{SG}_0}) \leftarrow \mathsf{KGen}_{\mathsf{SG}}(\mathsf{pp}, \mathsf{ID}_{\mathsf{SG}_0})$, $(\mathsf{pk}_{\mathsf{SG}_1}, \mathsf{sk}_{\mathsf{SG}_1}) \leftarrow \mathsf{KGen}_{\mathsf{SG}}(\mathsf{pp}, \mathsf{ID}_{\mathsf{SG}_1})$, $(\mathsf{pk}_{\mathsf{R}_0}, \mathsf{sk}_{\mathsf{R}_0}) \leftarrow \mathsf{KGen}_{\mathsf{R}}(\mathsf{pp}, \mathsf{ID}_{\mathsf{R}_0})$, $(\mathsf{pk}_{\mathsf{R}_1}, \mathsf{sk}_{\mathsf{R}_1}) \leftarrow \mathsf{KGen}_{\mathsf{R}}(\mathsf{pp}, \mathsf{ID}_{\mathsf{R}_1})$ *and sends* $\mathsf{pp}, \mathsf{pk}_{\mathsf{SG}_0}, \mathsf{pk}_{\mathsf{SG}_1}, \mathsf{pk}_{\mathsf{R}_0}, \mathsf{pk}_{\mathsf{R}_1}$ *to* $\mathcal{A}$*. Then,* $\mathcal{A}$ *outputs* $(i, j, kw, i', j', kw') \in (\{0,1\}^2 \times \mathcal{KW})^2$*.* $\mathcal{C}$ *runs* $\mathsf{C}_{kw} \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{pk}_{\mathsf{SG}_i}, \mathsf{pk}_{\mathsf{R}_j}, \mathsf{sk}_{\mathsf{SG}_i}, kw)$ *and* $\mathsf{T}_{kw'} \leftarrow \mathsf{Trapdoor}(\mathsf{pp}, \mathsf{pk}_{\mathsf{R}_{j'}}, \mathsf{pk}_{\mathsf{SG}_{i'}}, \mathsf{sk}_{\mathsf{R}_{j'}}, kw')$*.* $\mathcal{A}$ *wins if* $1 \leftarrow \mathsf{Test}(\mathsf{C}_{kw}, \mathsf{T}_{kw'})$ *and* $(i, j, kw) \neq (i', j', kw')$ *hold. The advantage of* $\mathcal{A}$ *is defined as*

$$\mathsf{Adv}^{\mathsf{consist}}_{\mathsf{PAEKS},\mathcal{A}}(\lambda) = \Pr[\mathcal{A} \textit{ wins}]$$

We say that the PAEKS scheme is consistent if for all PPT adversaries, $\mathsf{Adv}^{\mathsf{consist}}_{\mathsf{PAEKS},\mathcal{A}}(\lambda)$ *is negligible in* λ*.*

Next, we define full ciphertext indistinguishability against insiders (insider full CI).[2] This notion guarantees that no information about keywords is revealed from ciphertexts, even against insiders, i.e., group members of the sender group. To capture security against insiders, the adversary is given access to the secret keys of sender group members. Additionally, "full" security means that the adversary is allowed to obtain ciphertexts of the challenge keywords. As in the Li-Boyen definition [26], an adversary $\mathcal{A}$ is allowed to issue trapdoor queries. We remark that in [26], $\mathcal{A}$ submits $(\mathsf{pk}_{\mathsf{S}}, kw)$, where pk_{S} is a sender's public key chosen by $\mathcal{A}$, whereas in our definition, $\mathcal{A}$ submits $(\mathsf{ID}_{\mathsf{SG}}, kw)$. At first glance, allowing pk_{S} as input may seem to capture the case where an adversarially generated public key is used for trapdoor generation. However, in their security proof, if $\mathsf{S} \notin \mathcal{L}$ (i.e., the NIKE public key for S has not been generated), the simulator generates $(\mathsf{pk}_{\mathsf{S}}, \mathsf{sk}_{\mathsf{S}}) \leftarrow \mathsf{NIKE.Gen}(\mathsf{S})$ and uses pk_{S}. Thus, HKR-CKS security suffices. To capture this behavior more explicitly, in our definition the oracle takes $\mathsf{ID}_{\mathsf{SG}}$ instead of $\mathsf{pk}_{\mathsf{SG}}$ as input. We also remark that the encryption oracle is not defined because $\mathcal{A}$ is allowed to posses all sender group's secret keys, and can generate ciphertexts without accessing the encryption oracle. In addition, $\mathcal{A}$ indicates a

[2] In PEKS, since an adversary can obtain ciphertexts for arbitrary keywords using the receiver's public key, allowing encryption queries for the challenge keyword, namely, full security, should be regarded as a standard and minimal security requirement under the CI setting.

receiver as an encryption query (See the definition of outsider TI). Thus, we remove the $\mathsf{KeyGenR}$ oracle since it is redundant.

Definition 12 (Full Ciphertext Indistinguishability against Insiders (Insider Full CI))**.**

Setup: *$\mathcal{C}$ runs* $\mathsf{pp} \leftarrow \mathsf{SetUp}(1^\lambda), (\mathsf{pk}_{\mathsf{R}^*}, \mathsf{sk}_{\mathsf{R}^*}) \leftarrow \mathsf{KGen}_{\mathsf{R}}(\mathsf{pp}, \mathsf{ID}_{\mathsf{R}^*})$ *and send* $\mathsf{pp}, \mathsf{pk}_{\mathsf{R}^*}$ *to $\mathcal{A}$. $\mathcal{C}$ defines the oracle lists* $\mathcal{L}_{\mathsf{KGCorr}} \leftarrow \emptyset$ *and* $\mathcal{L}_{\mathsf{Trapdoor}} \leftarrow \emptyset$.

Phase I: *The following oracles are available to $\mathcal{A}$.*

KeyGenSG and Corrupt oracle: *$\mathcal{A}$ sends $\mathcal{C}$* $\mathsf{ID}_{\mathsf{SG}}$ *where* $(\mathsf{ID}_{\mathsf{SG}}, \cdot, \cdot) \notin \mathcal{L}_{\mathsf{KGCorr}}$. *$\mathcal{C}$ returns* $(\mathsf{pk}_{\mathsf{SG}}, \mathsf{sk}_{\mathsf{SG}}) \leftarrow \mathsf{KGen}_{\mathsf{SG}}(\mathsf{pp}, \mathsf{ID}_{\mathsf{SG}})$ *and records the tuple* $\mathcal{L}_{\mathsf{KGCorr}} \leftarrow \mathcal{L}_{\mathsf{KGCorr}} \cup \{(\mathsf{ID}_{\mathsf{SG}}, \mathsf{pk}_{\mathsf{SG}}, \mathsf{sk}_{\mathsf{SG}})\}$.

Trapdoor oracle: *$\mathcal{A}$ sends $\mathcal{C}$* $(\mathsf{ID}_{\mathsf{SG}}, kw)$ *where* $(\mathsf{ID}_{\mathsf{SG}}, \cdot, \cdot) \in \mathcal{L}_{\mathsf{KGCorr}}$. *$\mathcal{C}$ retrieves* $(\mathsf{ID}_{\mathsf{SG}}, \mathsf{pk}_{\mathsf{SG}}, \mathsf{sk}_{\mathsf{SG}})$ *from* $\mathcal{L}_{\mathsf{KGCorr}}$ *and returns* $\mathsf{T}_{kw} \leftarrow \mathsf{Trapdoor}(\mathsf{pp}, \mathsf{pk}_{\mathsf{R}^*}, \mathsf{pk}_{\mathsf{SG}}, \mathsf{sk}_{\mathsf{R}^*}, kw)$, *then records* $\mathcal{L}_{\mathsf{Trapdoor}} \leftarrow \mathcal{L}_{\mathsf{Trapdoor}} \cup \{(\mathsf{ID}_{\mathsf{SG}}, kw)\}$.

Challenge Phase: *$\mathcal{A}$ sends $\mathcal{C}$ the identity of the challenge sender group and two challenge keywords* $(\mathsf{ID}_{\mathsf{SG}^*}, kw_0^*, kw_1^*)$ *where* $(\mathsf{ID}_{\mathsf{SG}^*}, \cdot, \cdot) \in \mathcal{L}_{\mathsf{KGCorr}}$, $(\mathsf{ID}_{\mathsf{SG}^*}, kw_0^*) \notin \mathcal{L}_{\mathsf{Trapdoor}}$, *and* $(\mathsf{ID}_{\mathsf{SG}^*}, kw_1^*) \notin \mathcal{L}_{\mathsf{Trapdoor}}$. *$\mathcal{C}$ chooses* $b \xleftarrow{\$} \{0,1\}$ *and returns the challenge ciphertext* $\mathsf{C}_{kw_b^*} \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{pk}_{\mathsf{R}^*}, \mathsf{pk}_{\mathsf{SG}^*}, \mathsf{sk}_{\mathsf{SG}^*}, kw_b^*)$.

Phase II: *The following oracles are available to $\mathcal{A}$.*

KeyGenSG and Corrupt oracle: *$\mathcal{A}$ sends $\mathcal{C}$* $\mathsf{ID}_{\mathsf{SG}}$ *where* $(\mathsf{ID}_{\mathsf{SG}}, \cdot, \cdot) \in \mathcal{L}_{\mathsf{KGCorr}}$. *$\mathcal{C}$ returns* $(\mathsf{pk}_{\mathsf{SG}}, \mathsf{sk}_{\mathsf{SG}}) \leftarrow \mathsf{KGen}_{\mathsf{SG}}(\mathsf{pp}, \mathsf{ID}_{\mathsf{SG}})$ *and records the tuple* $\mathcal{L}_{\mathsf{KGCorr}} \leftarrow \mathcal{L}_{\mathsf{KGCorr}} \cup \{(\mathsf{ID}_{\mathsf{SG}}, \mathsf{pk}_{\mathsf{SG}}, \mathsf{sk}_{\mathsf{SG}})\}$.

Trapdoor oracle: *$\mathcal{A}$ sends $\mathcal{C}$* $(\mathsf{ID}_{\mathsf{SG}}, kw)$ *where* $(\mathsf{ID}_{\mathsf{SG}}, \cdot, \cdot) \in \mathcal{L}_{\mathsf{KGCorr}}$, $(\mathsf{ID}_{\mathsf{SG}}, kw) \neq (\mathsf{ID}_{\mathsf{SG}^*}, kw_0^*)$ *and* $(\mathsf{ID}_{\mathsf{SG}}, kw) \neq (\mathsf{ID}_{\mathsf{SG}^*}, kw_1^*)$. *$\mathcal{C}$ retrieves* $(\mathsf{ID}_{\mathsf{SG}}, \mathsf{pk}_{\mathsf{SG}}, \mathsf{sk}_{\mathsf{SG}})$ *from* $\mathcal{L}_{\mathsf{KGCorr}}$ *and returns* $\mathsf{T}_{kw} \leftarrow \mathsf{Trapdoor}(\mathsf{pp}, \mathsf{pk}_{\mathsf{R}^*}, \mathsf{pk}_{\mathsf{SG}}, \mathsf{sk}_{\mathsf{R}^*}, kw)$, *then records* $\mathcal{L}_{\mathsf{Trapdoor}} \leftarrow \mathcal{L}_{\mathsf{Trapdoor}} \cup \{(\mathsf{ID}_{\mathsf{SG}}, kw)\}$.

Guess: *$\mathcal{A}$ outputs* $b' \in \{0,1\}$. *The advantage of $\mathcal{A}$ is defined as*

$$\mathsf{Adv}^{\mathsf{InsiderFullCI}}_{\mathsf{PAEKS},\mathcal{A}}(\lambda) = |\Pr[b = b'] - 1/2|$$

We say that the PAEKS scheme provides insider full ciphertext indistinguishability if for all PPT adversaries, $\mathsf{Adv}^{\mathsf{InsiderFullCI}}_{\mathsf{PAEKS},\mathcal{A}}(\lambda)$ *is negligible in* λ.

Here, we define trapdoor indistinguishability against outsiders (outsider TI), which guarantees that no information about keywords is revealed from trapdoors against outsiders. This model does not allow the adversary to obtain trapdoors of the challenge keywords through the trapdoor oracle. In contrast to CI, where full security is regarded as a standard requirement, TI is often considered in a more relaxed form. We remark that, in the case of TI, linkability of trapdoors is allowed, which may lead to the leakage of whether the same keyword has been queried multiple times. From this perspective, achieving full TI is clearly desirable. However, achieving full TI often incurs additional costs. For instance, in the case of the Li-Boyen generic construction [26], it is currently unclear how to obtain discrete-logarithm-based pairing-free instantiations that satisfy full TI. Moreover, unlike searchable ciphertexts, trapdoors are not persistently stored but are typically ephemeral objects

generated for query purposes. Taking these observations into account, even TI without full security still provides meaningful protection in practice and thus has inherent significance. Regarding the encryption oracle, $\mathcal{A}$ sends $(\mathsf{ID_{SG}}, \mathsf{ID_R}, kw)$. Here, what $\mathcal{A}$ cannot perform on its own is to execute $\mathsf{Enc}(\mathsf{pp}, \mathsf{pk_{SG^*}}, \mathsf{pk_R}, \mathsf{sk_{SG^*}}, kw)$. Thus, it may appear sufficient for $\mathcal{A}$ to send $(\mathsf{ID_R}, kw)$ and obtain $\mathsf{Enc}(\mathsf{pp}, \mathsf{pk_{SG^*}}, \mathsf{pk_R}, \mathsf{sk_{SG^*}}, kw)$. However, in our definition, $\mathcal{A}$ specifies $\mathsf{ID_{SG^*}}$ in the challenge phase. To capture the case where $\mathcal{A}$ issues encryption queries from a sender group that has not been corrupted (including the challenge sender group), we require $\mathcal{A}$ to send $(\mathsf{ID_{SG}}, \mathsf{ID_R}, kw)$ in our definition.

Definition 13 (Trapdoor Indistinguishability against Outsiders (Outsider TI))**.**

Setup: *$\mathcal{C}$ runs* $\mathsf{pp} \leftarrow \mathsf{SetUp}(1^\lambda), (\mathsf{pk_{R^*}}, \mathsf{sk_{R^*}}) \leftarrow \mathsf{KGen_R}(\mathsf{pp}, \mathsf{ID_{R^*}})$ *and send* $\mathsf{pp}, \mathsf{pk_{R^*}}$ *to $\mathcal{A}$. $\mathcal{C}$ defines the oracle lists* $\mathcal{L}_{\mathsf{KeyGenSG}} \leftarrow \emptyset$, $\mathcal{L}_{\mathsf{CorruptSG}} \leftarrow \emptyset$, $\mathcal{L}_{\mathsf{KeyGenR}} \leftarrow \emptyset$, $\mathcal{L}_{\mathsf{Enc}} \leftarrow \emptyset$ *and* $\mathcal{L}_{\mathsf{Trapdoor}} \leftarrow \emptyset$.

Phase I: *The following oracles are available to $\mathcal{A}$.*

- **KeyGenSG Oracle:** *$\mathcal{A}$ sends $\mathcal{C}$* $\mathsf{ID_{SG}}$ *where* $(\mathsf{ID_{SG}}, \cdot, \cdot) \notin \mathcal{L}_{\mathsf{KeyGenSG}}$. *$\mathcal{C}$ runs* $(\mathsf{pk_{SG}}, \mathsf{sk_{SG}}) \leftarrow \mathsf{KGen_{SG}}(\mathsf{pp}, \mathsf{ID_{SG}})$ *and returns* $\mathsf{pk_{SG}}$ *then records* $\mathcal{L}_{\mathsf{KeyGenSG}} \leftarrow \mathcal{L}_{\mathsf{KeyGenSG}} \cup \{(\mathsf{ID_{SG}}, \mathsf{pk_{SG}}, \mathsf{sk_{SG}})\}$.
- **CorruptSG Oracle:** *$\mathcal{A}$ sends $\mathcal{C}$* $\mathsf{ID_{SG}}$ *where* $(\mathsf{ID_{SG}}, \cdot, \cdot) \in \mathcal{L}_{\mathsf{KeyGenSG}}$. *$\mathcal{C}$ retrieves* $(\mathsf{ID_{SG}}, \mathsf{pk_{SG}}, \mathsf{sk_{SG}}) \in \mathcal{L}_{\mathsf{KeyGenSG}}$ *and returns* $\mathsf{sk_{SG}}$ *then records the identity* $\mathcal{L}_{\mathsf{CorruptSG}} \leftarrow \mathcal{L}_{\mathsf{CorruptSG}} \cup \{\mathsf{ID_{SG}}\}$.
- **KeyGenR Oracle:** *$\mathcal{A}$ sends $\mathcal{C}$* $\mathsf{ID_R}$ *where* $(\mathsf{ID_R}, \cdot, \cdot) \notin \mathcal{L}_{\mathsf{KeyGenR}}$. *$\mathcal{C}$ runs* $(\mathsf{pk_R}, \mathsf{sk_R}) \leftarrow \mathsf{KGen_R}(\mathsf{pp}, \mathsf{ID_R})$ *and returns* $\mathsf{pk_R}$ *then records* $\mathcal{L}_{\mathsf{KeyGenR}} \leftarrow \mathcal{L}_{\mathsf{KeyGenR}} \cup \{(\mathsf{ID_R}, \mathsf{pk_R}, \mathsf{sk_R})\}$.
- **Encryption Oracle:** *$\mathcal{A}$ sends $\mathcal{C}$* $(\mathsf{ID_{SG}}, \mathsf{ID_R}, kw)$ *where* $(\mathsf{ID_{SG}}, \cdot, \cdot) \in \mathcal{L}_{\mathsf{KeyGenSG}}$, $(\mathsf{ID_R}, \cdot, \cdot) \in \mathcal{L}_{\mathsf{KeyGenR}}$. *$\mathcal{C}$ retrieves* $(\mathsf{ID_{SG}}, \mathsf{pk_{SG}}, \mathsf{sk_{SG}}) \in \mathcal{L}_{\mathsf{KeyGenSG}}$ *and* $(\mathsf{ID_R}, \mathsf{pk_R}, \mathsf{sk_R}) \in \mathcal{L}_{\mathsf{KeyGenR}}$ *then runs* $\mathsf{C}_{kw} \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{pk_{SG}}, \mathsf{pk_R}, \mathsf{sk_{SG}}, kw)$. *$\mathcal{C}$ returns C_kw and records* $\mathcal{L}_{\mathsf{Enc}} \leftarrow \mathcal{L}_{\mathsf{Enc}} \cup \{(\mathsf{ID_{SG}}, \mathsf{ID_R}, kw)\}$
- **Trapdoor Oracle:** *$\mathcal{A}$ sends $\mathcal{C}$* $(\mathsf{ID_{SG}}, kw)$ *where* $(\mathsf{ID_{SG}}, \cdot, \cdot) \in \mathcal{L}_{\mathsf{KeyGenSG}}$. *$\mathcal{C}$ retrieves* $(\mathsf{ID_{SG}}, \mathsf{pk_{SG}}, \mathsf{sk_{SG}}) \in \mathcal{L}_{\mathsf{KeyGenSG}}$ *and returns* $\mathsf{T}_{kw} \leftarrow \mathsf{Trapdoor}(\mathsf{pp}, \mathsf{pk_{R^*}}, \mathsf{pk_{SG}}, \mathsf{sk_{R^*}}, kw)$ *then records* $\mathcal{L}_{\mathsf{Trapdoor}} \leftarrow \mathcal{L}_{\mathsf{Trapdoor}} \cup \{(\mathsf{ID_{SG}}, kw)\}$.

Challenge Phase: *$\mathcal{A}$ sends $\mathcal{C}$ the identity of the challenge sender group and two challenge keywords* $(\mathsf{ID_{SG^*}}, kw_0^*, kw_1^*)$ *where* $(\mathsf{ID_{SG^*}}, \cdot, \cdot) \in \mathcal{L}_{\mathsf{KeyGenSG}}$, $\mathsf{ID_{SG^*}} \notin \mathcal{L}_{\mathsf{CorruptSG}}$, $(\mathsf{ID_{SG^*}}, \mathsf{ID_{R^*}} kw_0^*) \notin \mathcal{L}_{\mathsf{Enc}}$, $(\mathsf{ID_{SG^*}}, \mathsf{ID_{R^*}}, kw_1^*,) \notin \mathcal{L}_{\mathsf{Enc}}$, $(\mathsf{ID_{SG^*}}, kw_0^*) \notin \mathcal{L}_{\mathsf{Trapdoor}}$ *and* $((\mathsf{ID_{SG^*}}, kw_1^*) \notin \mathcal{L}_{\mathsf{Trapdoor}}$. *$\mathcal{C}$ chooses* $b \xleftarrow{\$} \{0,1\}$ *and returns the challenge trapdoor* $\mathsf{T}_{kw_b^*} \leftarrow \mathsf{Trapdoor}(\mathsf{pp}, \mathsf{pk_{R^*}}, \mathsf{pk_{SG^*}}, \mathsf{sk_{R^*}}, kw_b^*)$.

Phase II: *The following oracles are available to $\mathcal{A}$.*

- **KeyGenSG Oracle:** *$\mathcal{A}$ sends $\mathcal{C}$* $\mathsf{ID_{SG}}$ *where* $(\mathsf{ID_{SG}}, \cdot, \cdot) \notin \mathcal{L}_{\mathsf{KeyGenSG}}$. *$\mathcal{C}$ runs* $(\mathsf{pk_{SG}}, \mathsf{sk_{SG}}) \leftarrow \mathsf{KGen_{SG}}(\mathsf{pp}, \mathsf{ID_{SG}})$ *and returns* $\mathsf{pk_{SG}}$ *then records* $\mathcal{L}_{\mathsf{KeyGenSG}} \leftarrow \mathcal{L}_{\mathsf{KeyGenSG}} \cup \{(\mathsf{ID_{SG}}, \mathsf{pk_{SG}}, \mathsf{sk_{SG}})\}$.
- **CorruptSG Oracle:** *$\mathcal{A}$ sends $\mathcal{C}$* $\mathsf{ID_{SG}}$ *where* $(\mathsf{ID_{SG}}, \cdot, \cdot) \in \mathcal{L}_{\mathsf{KeyGenSG}}$, $\mathsf{ID_{SG}} \neq \mathsf{ID_{SG^*}}$. *$\mathcal{C}$ retrieves* $(\mathsf{ID_{SG}}, \mathsf{pk_{SG}}, \mathsf{sk_{SG}}) \in \mathcal{L}_{\mathsf{KeyGenSG}}$ *and returns* $\mathsf{sk_{SG}}$ *then records the identity* $\mathcal{L}_{\mathsf{CorruptSG}} \leftarrow \mathcal{L}_{\mathsf{CorruptSG}} \cup \{\mathsf{ID_{SG}}\}$.
- **KeyGenR Oracle:** *$\mathcal{A}$ sends $\mathcal{C}$* $\mathsf{ID_R}$ *where* $(\mathsf{ID_R}, \cdot, \cdot) \notin \mathcal{L}_{\mathsf{KeyGenR}}$. *$\mathcal{C}$ runs* $(\mathsf{pk_R}, \mathsf{sk_R}) \leftarrow \mathsf{KGen_R}(\mathsf{pp}, \mathsf{ID_R})$ *and returns* $\mathsf{pk_R}$ *then records* $\mathcal{L}_{\mathsf{KeyGenR}} \leftarrow \mathcal{L}_{\mathsf{KeyGenR}} \cup \{(\mathsf{ID_R}, \mathsf{pk_R}, \mathsf{sk_R})\}$.

Encryption Oracle: $\mathcal{A}$ *sends* $\mathcal{C}$ $(\mathsf{ID_{SG}}, \mathsf{ID_R}, kw)$ *where* $(\mathsf{ID_{SG}}, \cdot, \cdot) \in \mathcal{L}_{\mathsf{KeyGenSG}}$, $(\mathsf{ID_R}, \cdot, \cdot) \in \mathcal{L}_{\mathsf{KeyGenR}}$. $(\mathsf{ID_{SG}}, \mathsf{ID_R}, kw) \neq (\mathsf{ID_{SG^*}}, \mathsf{ID_{R^*}}, kw_0^*)$ *and* $(\mathsf{ID_{SG}}, \mathsf{ID_R}, kw) \neq (\mathsf{ID_{SG^*}}, \mathsf{ID_{R^*}}, kw_1^*)$. $\mathcal{C}$ *retrieves* $(\mathsf{ID_{SG}}, \mathsf{pk_{SG}}, \mathsf{sk_{SG}}) \in \mathcal{L}_{\mathsf{KeyGenSG}}$ *and* $(\mathsf{ID_R}, \mathsf{pk_R}, \mathsf{sk_R}) \in \mathcal{L}_{\mathsf{KeyGenR}}$ *then runs* $\mathsf{C}_{kw} \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{pk_{SG}}, \mathsf{pk_R}, \mathsf{sk_{SG}}, kw)$. $\mathcal{C}$ *returns* C_kw *and records* $\mathcal{L}_{\mathsf{Enc}} \leftarrow \mathcal{L}_{\mathsf{Enc}} \cup \{(\mathsf{ID_{SG}}, \mathsf{ID_R}, kw)\}$.

Trapdoor Oracle: $\mathcal{A}$ *sends* $\mathcal{C}$ $(\mathsf{ID_{SG}}, kw)$ *where* $(\mathsf{ID_{SG}}, \cdot, \cdot) \in \mathcal{L}_{\mathsf{KeyGenSG}}$, $(\mathsf{ID_{SG}}, kw) \neq (\mathsf{ID_{SG^*}}, kw_0^*)$ *and* $(\mathsf{ID_{SG}}, kw) \neq (\mathsf{ID_{SG^*}}, kw_1^*)$. $\mathcal{C}$ *retrieves* $(\mathsf{ID_{SG}}, \mathsf{pk_{SG}}, \mathsf{sk_{SG}}) \in \mathcal{L}_{\mathsf{KeyGenSG}}$ *and returns* $\mathsf{T}_{kw} \leftarrow \mathsf{Trapdoor}(\mathsf{pp}, \mathsf{pk_{R^*}}, \mathsf{pk_{SG}}, \mathsf{sk_{R^*}}, kw)$ *then records* $\mathcal{L}_{\mathsf{Trapdoor}} \leftarrow \mathcal{L}_{\mathsf{Trapdoor}} \cup \{(\mathsf{ID_{SG}}, kw)\}$.

Guess: $\mathcal{A}$ *outputs* $b' \in \{0,1\}$. *The advantage of* $\mathcal{A}$ *is defined as*

$$\mathsf{Adv}^{\mathsf{OutsiderTI}}_{\mathsf{PAEKS},\mathcal{A}}(\lambda) = |\Pr[b = b'] - 1/2|$$

We say that the PAEKS scheme provides outsider trapdoor indistinguishability if for all PPT adversaries, $\mathsf{Adv}^{\mathsf{OutsiderTI}}_{\mathsf{PAEKS},\mathcal{A}}(\lambda)$ *is negligible in* λ.

4 Our Group-Oriented PAEKS Construction

Proposed Generic Construction. We construct $\mathsf{PAEKS} = (\mathsf{SetUp}, \mathsf{KGen_{SG}}, \mathsf{KGen_R}, \mathsf{Enc}, \mathsf{Trapdoor}, \mathsf{Test})$ from $\mathsf{NIKE} = (\mathsf{NIKE.SetUp}, \mathsf{NIKE.Gen}, \mathsf{NIKE.SharedKey})$, PRP, and $\mathsf{PEKS} = (\mathsf{PEKS.SetUp}, \mathsf{PEKS.KGen}, \mathsf{PEKS.Enc}, \mathsf{PEKS.Trapdoor}, \mathsf{PEKS.Test})$ as follows. As mentioned in the technical overview section, A shared NIKE key $\mathsf{K_{NIKE}}$ established between a sender group and a receiver is then used as the key of PRP F, and each keyword kw is transformed into the pseudo-random keyword $\mathsf{pr}\text{-}kw \leftarrow F_{\mathsf{K_{NIKE}}}(kw)$. Here, we assume that $\mathcal{KW} := \{0,1\}^{\ell(\lambda)}$. Then, the sender encrypts $\mathsf{pr}\text{-}kw$ using the $\mathsf{PEKS.Enc}$ algorithm, and the receiver generates a trapdoor for $\mathsf{pr}\text{-}kw$ using the $\mathsf{PEKS.Trapdoor}$ algorithm, respectively. The test algorithm is the same as the $\mathsf{PEKS.Test}$ algorithm.

$\mathsf{SetUp}(1^\lambda)$**:** Run $\mathsf{pp_{NIKE}} \leftarrow \mathsf{NIKE.SetUp}(1^\lambda)$ and $\mathsf{pp_{PEKS}} \leftarrow \mathsf{PEKS.SetUp}(1^\lambda)$. Outputs $\mathsf{pp} = (\mathsf{pp_{NIKE}}, \mathsf{pp_{PEKS}})$.

$\mathsf{KGen_{SG}}(\mathsf{pp}, \mathsf{ID_{SG}})$**:** Parse $\mathsf{pp} = (\mathsf{pp_{NIKE}}, \mathsf{pp_{PEKS}})$ and set $\mathsf{ID_{SG}} = \mathsf{NIKE.ID_{SG}}$. Run $(\mathsf{NIKE.pk_{SG}}, \mathsf{NIKE.sk_{SG}}) \leftarrow \mathsf{NIKE.Gen}(\mathsf{NIKE.ID_{SG}})$. Output $(\mathsf{pk_{SG}}, \mathsf{sk_{SG}}) = (\mathsf{NIKE.pk_{SG}}, \mathsf{NIKE.sk_{SG}})$.

$\mathsf{KGen_R}(\mathsf{pp}, \mathsf{ID_R})$**:** Parse $\mathsf{pp} = (\mathsf{pp_{NIKE}}, \mathsf{pp_{PEKS}})$ and $\mathsf{ID_R} = \mathsf{NIKE.ID_R}$. Run $(\mathsf{pk_{PEKS}}, \mathsf{sk_{PEKS}}) \leftarrow \mathsf{PEKS.KGen}(\mathsf{pp_{PEKS}})$ and $(\mathsf{NIKE.pk_R}, \mathsf{NIKE.sk_R}) \leftarrow \mathsf{NIKE.Gen}(\mathsf{NIKE.ID_R})$. Set $\mathsf{pk_R} = (\mathsf{pk_{PEKS}}, \mathsf{NIKE.pk_R})$ and $\mathsf{sk_R} = (\mathsf{sk_{PEKS}}, \mathsf{NIKE.sk_R})$. Output $(\mathsf{pk_R}, \mathsf{sk_R})$.

$\mathsf{Enc}(\mathsf{pp}, \mathsf{pk_{SG}}, \mathsf{pk_R}, \mathsf{sk_{SG}}, kw)$**:** Parse $\mathsf{pp} = (\mathsf{pp_{NIKE}}, \mathsf{pp_{PEKS}})$, $(\mathsf{pk_{SG}}, \mathsf{sk_{SG}}) = (\mathsf{NIKE.pk_{SG}}, \mathsf{NIKE.sk_{SG}})$ and $\mathsf{pk_R} = (\mathsf{pk_{PEKS}}, \mathsf{NIKE.pk_R})$. Run $\mathsf{K_{NIKE}} \leftarrow \mathsf{NIKE.SharedKey}(\mathsf{NIKE.ID_R}, \mathsf{NIKE.pk_R}, \mathsf{NIKE.ID_{SG}}, \mathsf{NIKE.sk_{SG}})$, $\mathsf{pr}\text{-}kw \leftarrow F_{\mathsf{K_{NIKE}}}(kw)$ and $\mathsf{C}_{\mathsf{PEKS},\mathsf{pr}\text{-}kw} \leftarrow \mathsf{PEKS.Enc}(\mathsf{pp_{PEKS}}, \mathsf{pk_{PEKS}}, \mathsf{pr}\text{-}kw)$. Output $\mathsf{C}_{kw} = \mathsf{C}_{\mathsf{PEKS},\mathsf{pr}\text{-}kw}$.

$\mathsf{Trapdoor}(\mathsf{pp}, \mathsf{pk_R}, \mathsf{pk_{SG}}, \mathsf{sk_R}, kw)$**:** Parse $\mathsf{pp} = (\mathsf{pp_{NIKE}}, \mathsf{pp_{PEKS}})$, $\mathsf{pk_R} = (\mathsf{pk_{PEKS}}, \mathsf{NIKE.pk_R})$, $\mathsf{sk_R} = (\mathsf{sk_{PEKS}}, \mathsf{NIKE.sk_R})$ and $\mathsf{pk_{SG}} = \mathsf{NIKE.pk_{SG}}$. Run $\mathsf{K_{NIKE}} \leftarrow \mathsf{NIKE.SharedKey}(\mathsf{NIKE.ID_{SG}}, \mathsf{NIKE.pk_{SG}}, \mathsf{NIKE.ID_R}, \mathsf{NIKE.sk_R})$, $\mathsf{pr}\text{-}kw \leftarrow F_{\mathsf{K_{NIKE}}}(kw)$ and $\mathsf{T}_{\mathsf{PEKS},\mathsf{pr}\text{-}kw} \leftarrow \mathsf{PEKS.Trapdoor}(\mathsf{pp_{PEKS}}, \mathsf{sk_{PEKS}}, \mathsf{pr}\text{-}kw)$. Output $\mathsf{T}_{kw} = \mathsf{T}_{\mathsf{PEKS},\mathsf{pr}\text{-}kw}$.

$\mathsf{Test}(\mathsf{C}_{kw}, \mathsf{T}_{kw})$**:** Parse $\mathsf{C}_{kw} = \mathsf{C}_{\mathsf{PEKS},\mathsf{pr}\text{-}kw}$ and $\mathsf{T}_{kw} = \mathsf{T}_{\mathsf{PEKS},\mathsf{pr}\text{-}kw}$. Output the result of $\mathsf{PEKS.Test}$ $(\mathsf{C}_{\mathsf{PEKS},\mathsf{pr}\text{-}kw}, \mathsf{T}_{\mathsf{PEKS},\mathsf{pr}\text{-}kw})$.

Obviously, the PAEKS construction is correct if NIKE, PRP, and PEKS are correct.

5 Security Analysis

Theorem 1. *Our PAEKS construction is consistent if PEKS is consistent, NIKE is HKR-CKS secure, and PRP provides pseudo-randomness.*

Outline of the Security Proof. Our goal is to show that the consistency of PAEKS ultimately reduces to consistency of PEKS. To achieve this, for the output (i, j, kw, i', j', kw'), the converted $\mathsf{pr}\text{-}kw$, $\mathsf{pr}\text{-}kw'$ should satisfy $\mathsf{pr}\text{-}kw \neq \mathsf{pr}\text{-}kw'$. We replace $\mathsf{K}_{\mathsf{NIKE},i,j}$ and $\mathsf{K}_{\mathsf{NIKE},i',j'}$ to uniformly random key by using HKR-CKS, and then we replace $\mathsf{pr}\text{-}kw \leftarrow F_{\mathsf{K}_{\mathsf{NIKE},i,j}}(kw)$ and $\mathsf{pr}\text{-}kw' = F_{K_{\mathsf{NIKE},i',j'}}(kw')$ to truly random functions. The description of the game transition is provided as follows.

Proof. To prove the theorem, we define the sequences of games $\mathsf{Game}_0, \cdots, \mathsf{Game}_4$. Let $(i, j, kw, i', j', kw') \in (\{0,1\}^2 \times \mathcal{KW})^2$ be the output of $\mathcal{A}$.

Game_0**:** This is the same as the original game11. In this game, $\mathsf{K}_{\mathsf{NIKE},i,j} \leftarrow \mathsf{NIKE.SharedKey}(\mathsf{NIKE.ID}_{\mathsf{R}_j}, \mathsf{NIKE.pk}_{\mathsf{R}_j}, \mathsf{NIKE.ID}_{\mathsf{SG}_i}, \mathsf{NIKE.sk}_{\mathsf{SG}_i})$ and $\mathsf{K}_{\mathsf{NIKE},i',j'} \leftarrow \mathsf{NIKE.SharedKey}(\mathsf{NIKE.ID}_{\mathsf{R}_{j'}}, \mathsf{NIKE.pk}_{\mathsf{R}_{j'}}, \mathsf{NIKE.ID}_{\mathsf{SG}_{i'}}, \mathsf{NIKE.sk}_{\mathsf{SG}_{i'}})$, $\mathsf{pr}\text{-}kw = F_{\mathsf{K}_{\mathsf{NIKE},i,j}}(kw)$ and $\mathsf{pr}\text{-}kw' = F_{K_{\mathsf{NIKE},i',j'}}(kw')$.

Game_1**:** Game_1 is the same as Game_0 except that $\mathsf{K}_{\mathsf{NIKE},i,j}$ is randomly chosen from $\{0,1\}^\lambda$. We show that $|\Pr[\mathsf{Game}_0 = 1] - \Pr[\mathsf{Game}_1 = 1]|$ is negligible in λ if NIKE is HKR-CKS secure. Let $\mathcal{C}$ be the challenger of HKR-CKS game. We construct an algorithm $\mathcal{B}_1$ that breaks the HKR-CKS security using $\mathcal{A}$ as follows. For the output of $\mathcal{A}$ (i, j, kw, i', j', kw'), $\mathcal{B}_1$ sends $\mathcal{C}$ the test query $\mathsf{NIKE.ID}_{\mathsf{SG}_i}$ and $\mathsf{NIKE.ID}_{\mathsf{R}_j}$. $\mathcal{C}$ chooses $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, $\mathcal{C}$ returns $\mathsf{K}_{\mathsf{NIKE},0} \leftarrow \mathsf{NIKE.SharedKey}(\mathsf{NIKE.ID}_1, \mathsf{NIKE.pk}_i, \mathsf{ID}_j, \mathsf{NIKE.NIKE.sk}_j)$ and $\mathcal{B}_1$ simulates Game_0. Otherwise $b = 1$, $\mathcal{C}$ returns a random key $\mathsf{K}_{\mathsf{NIKE},1} \leftarrow \{0,1\}^\lambda$ and $\mathcal{B}_1$ simulates Game_1. Thus,

$$|\Pr[\mathsf{Game}_0 = 1] - \Pr[\mathsf{Game}_1 = 1]| \leqq \mathsf{Adv}^{\mathsf{HKR\text{-}CKS}}_{\mathsf{NIKE},\mathcal{B}_1}(\lambda)$$

holds.

Game_2**:** Game_2 is the same as Game_1 except that $\mathsf{K}_{\mathsf{NIKE},i',j'}$ is randomly chosen from $\{0,1\}^\lambda$. As in Game_1, $|\Pr[\mathsf{Game}_1 = 1] - \Pr[\mathsf{Game}_2 = 1]|$ is negligible in λ. Concretely, we can construct an algorithm $\mathcal{B}_2$ that breaks the HKR-CKS security using $\mathcal{A}$, and thus

$$|\Pr[\mathsf{Game}_1 = 1] - \Pr[\mathsf{Game}_2 = 1]| \leqq \mathsf{Adv}^{\mathsf{HKR\text{-}CKS}}_{\mathsf{NIKE},\mathcal{B}_2}(\lambda)$$

holds.

Game_3**:** Game_3 is the same as Game_2 except $\mathsf{pr}\text{-}kw$ is randomly chosen such that $\mathsf{pr}\text{-}kw \leftarrow \{0,1\}^{\ell(\lambda)}$. We show that $|\Pr[\mathsf{Game}_2 = 1] - \Pr[\mathsf{Game}_3 = 1]|$ is negligible in λ if the PRP provides pseudo-randomness. Let $\mathcal{C}$ be the challenger of pseudo-randomness game. We construct an algorithm $\mathcal{B}_3$ that breaks the pseudo-randomness using $\mathcal{A}$ as follows. $\mathcal{C}$ randomly chooses a PRP key $\mathsf{K}_{\mathsf{NIKE},i,j} \leftarrow \{0,1\}^\lambda$. $\mathcal{B}_3$ sends kw to $\mathcal{C}$. $\mathcal{C}$ returns either $\mathsf{pr}\text{-}kw \leftarrow F_{\mathsf{K}_{\mathsf{NIKE},i,j}}(kw)$ or $\mathsf{pr}\text{-}kw \xleftarrow{\$}$

$\{0,1\}^{\ell(\lambda)}$. If $\mathsf{pr}\text{-}kw = F_{\mathsf{K}_{\mathsf{NIKE},i,j}}(kw)$, $\mathcal{B}_3$ simulates Game_2. Otherwise $\mathsf{pr}\text{-}kw$ is a random value, $\mathcal{B}_3$ simulates Game_3. Thus,

$$|\Pr[\mathsf{Game}_2 = 1] - \Pr[\mathsf{Game}_3 = 1]| \leqq \mathsf{Adv}^{\mathsf{Pseudo\text{-}random}}_{\mathsf{PRP},\mathcal{B}_3}(\lambda)$$

holds.

Game_4: Game_4 is the same as Game_3 except except $\mathsf{pr}\text{-}kw'$ is randomly chosen such that $\mathsf{pr}\text{-}kw' \leftarrow \{0,1\}^{\ell(\lambda)}$. As in Game_3, $|\Pr[\mathsf{Game}_3 = 1] - \Pr[\mathsf{Game}_4 = 1]|$ is negligible in λ. Concretely, we can construct an algorithm $\mathcal{B}_4$ that breaks the pseudo-randomness using $\mathcal{A}$, and thus

$$|\Pr[\mathsf{Game}_3 = 1] - \Pr[\mathsf{Game}_4 = 1]| \leqq \mathsf{Adv}^{\mathsf{Pseudo\text{-}random}}_{\mathsf{PRP},\mathcal{B}_4}(\lambda)$$

holds.

Finally, we show that $\Pr[\mathsf{Game}_4 = 1]$ is negligible if PEKS is consistent. Let $\mathcal{C}$ be the challenger of the consistency of PEKS. We construct the algorithm $\mathcal{B}$ that breaks consistency of PAEKS using $\mathcal{A}$ as follows. For the output of $\mathcal{A}$ (i, j, kw, i', j', kw'), (kw, kw') is randomly chosen $\mathsf{pr}\text{-}kw \leftarrow \{0,1\}^{\ell(\lambda)}$ and $\mathsf{pr}\text{-}kw' \leftarrow \{0,1\}^{\ell(\lambda)}$ regardless of i, j, i' and j'. Thus, the probability that $\mathsf{pr}\text{-}kw = \mathsf{pr}\text{-}kw'$ holds is negligible in λ. $\mathcal{B}$ sends $(\mathsf{pr}\text{-}kw, \mathsf{pr}\text{-}kw')$ to $\mathcal{C}$. If $\mathcal{A}$ breaks consistency in PAEKS, then $\mathcal{B}$ breaks consistency in PEKS. Thus,

$$|\Pr[\mathsf{Game}_4 = 1]| \leqq \mathsf{Adv}^{\mathsf{consist}}_{\mathsf{PEKS},\mathcal{B}}(\lambda)$$

holds.

Here, the advantage of consistency in the PAEKS is shown as follows.

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{consist}}_{\mathsf{PAEKS},\mathcal{A}}(\lambda) &= \Pr[\mathsf{Game}_0 = 1] \\
&= \Big[\sum_{i=0}^{3}\{\Pr[\mathsf{Game}_i = 1] - \Pr[\mathsf{Game}_{(i+1)} = 1]\}\Big] + \Pr[\mathsf{Game}_4 = 1] \\
&\leqq \Big[\sum_{i=0}^{3}|\Pr[\mathsf{Game}_i = 1] - \Pr[\mathsf{Game}_{(i+1)} = 1]|\Big] + \Pr[\mathsf{Game}_4 = 1] \\
&\leqq \Big[\sum_{i=1}^{2}\mathsf{Adv}^{\mathsf{HKR\text{-}CKS}}_{\mathsf{NIKE},\mathcal{B}_i}(\lambda)\Big] + \Big[\sum_{j=3}^{4}\mathsf{Adv}^{\mathsf{Pseudo\text{-}random}}_{\mathsf{PRP},\mathcal{B}_j}(\lambda)\Big] + \mathsf{Adv}^{\mathsf{consist}}_{\mathsf{PEKS},\mathcal{B}}(\lambda)
\end{aligned}$$

This concludes the proof. □

Theorem 2. *Our PAEKS construction is insider full CI secure if* PEKS *is IND-CKA secure.*

Outline of the Security Proof. Our goal is to show that the insider full CI of PAEKS reduces to the IND-CKA of PEKS. To achieve this, we construct the algorithm $\mathcal{B}$ that breaks IND-CKA using $\mathcal{A}$ that breaks insider full CI. For $(\mathsf{ID}_{\mathsf{SG}^*}, kw_0^*, kw_1^*)$ declared by $\mathcal{A}$ in the challenge phase, $\mathcal{B}$ generates the challenge keywords of IND-CKA $(\mathsf{pr}\text{-}kw_0^*, \mathsf{pr}\text{-}kw_1^*)$. If $\mathcal{A}$ breaks insider full CI security, then $\mathcal{B}$ can break IND-CKA security. We remark that if $\mathsf{pr}\text{-}kw_0^* = \mathsf{pr}\text{-}kw_1^*$, then $\mathsf{Adv}^{\mathsf{InsiderFullCI}}_{\mathsf{PAEKS},\mathcal{A}}(\lambda) = 0$ holds. However, this does not happen if $kw_0^* \neq kw_1^*$ since F is PRP in our construction, i.e., the permutation guarantees that if two inputs are different values, then the outputs are also different. Similarly, for $kw \notin \{kw_0^*, kw_1^*\}$, the corresponding $\mathsf{pr}\text{-}kw$ is required to be $\mathsf{pr}\text{-}kw \notin \{\mathsf{pr}\text{-}kw_0^*, \mathsf{pr}\text{-}kw_1^*\}$. This is the reason behind we require that even if two distinct PRP keys, K and K', are known, it is infeasible to find distinct inputs, kw and kw', where $F_K(kw) = F_{K'}(kw')$ holds. This allows us to simulate the trapdoor oracle.

Proof. Let $\mathcal{C}$ be the challenger of the IND-CKA of PEKS. We construct the algorithm $\mathcal{B}$ that breaks insider full CI of PAEKS using $\mathcal{A}$ as follows. $\mathcal{C}$ runs $\mathsf{pp}_{\mathsf{PEKS}} \leftarrow \mathsf{PEKS.SetUp}(1^\lambda)$ and $(\mathsf{pk}^*_{\mathsf{PEKS}}, \mathsf{sk}^*_{\mathsf{PEKS}}) \leftarrow \mathsf{PEKS.KGen}(\mathsf{pp}_{\mathsf{PEKS}})$, then sends $\mathsf{pp}_{\mathsf{PEKS}}$ and $\mathsf{pk}^*_{\mathsf{PEKS}}$ to $\mathcal{B}$. $\mathcal{B}$ runs $\mathsf{pp}_{\mathsf{NIKE}} \leftarrow \mathsf{NIKE.SetUp}(1^\lambda)$ and $(\mathsf{NIKE.pk}_{\mathsf{R}^*}, \mathsf{NIKE.sk}_{\mathsf{R}^*}) \leftarrow \mathsf{NIKE.Gen}(\mathsf{NIKE.ID}_{\mathsf{R}^*})$, then sends $\mathsf{pp} = (\mathsf{pp}_{\mathsf{PEKS}}, \mathsf{pp}_{\mathsf{NIKE}})$ and $\mathsf{pk}_{\mathsf{R}^*} = (\mathsf{pk}^*_{\mathsf{PEKS}}, \mathsf{NIKE.pk}_{\mathsf{R}^*})$ to $\mathcal{A}$.

KeyGenSG and Corrupt oracle: $\mathcal{A}$ sends $\mathsf{ID}_{\mathsf{SG}}$ where $(\mathsf{ID}_{\mathsf{SG}}, \cdot, \cdot) \notin \mathcal{L}_{\mathsf{KGCorr}}$. $\mathcal{B}$ sets $\mathsf{ID}_{\mathsf{SG}} = \mathsf{NIKE.ID}_{\mathsf{SG}}$, runs $(\mathsf{NIKE.pk}_{\mathsf{SG}}, \mathsf{NIKE.sk}_{\mathsf{SG}}) \leftarrow \mathsf{NIKE.Gen}(\mathsf{NIKE.ID}_{\mathsf{SG}})$, sets $(\mathsf{pk}_{\mathsf{SG}}, \mathsf{sk}_{\mathsf{SG}}) = (\mathsf{NIKE.pk}_{\mathsf{SG}}, \mathsf{NIKE.sk}_{\mathsf{SG}})$, records $\mathcal{L}_{\mathsf{KGCorr}} \leftarrow \mathcal{L}_{\mathsf{KGCorr}} \cup \{(\mathsf{ID}_{\mathsf{SG}}, \mathsf{pk}_{\mathsf{SG}}, \mathsf{sk}_{\mathsf{SG}})\}$, and returns $(\mathsf{pk}_{\mathsf{SG}}, \mathsf{sk}_{\mathsf{SG}})$ to $\mathcal{A}$.

Trapdoor oracle: $\mathcal{A}$ sends $(\mathsf{ID}_{\mathsf{SG}}, kw)$ where $(\mathsf{ID}_{\mathsf{SG}}, \cdot, \cdot) \in \mathcal{L}_{\mathsf{KGCorr}}$. $\mathcal{B}$ retrieves $(\mathsf{ID}_{\mathsf{SG}}, \mathsf{pk}_{\mathsf{SG}}, \mathsf{sk}_{\mathsf{SG}})$ from $\mathcal{L}_{\mathsf{KGCorr}}$ and compute $\mathsf{K}_{\mathsf{NIKE}} \leftarrow \mathsf{NIKE.SharedKey}(\mathsf{NIKE.ID}_{\mathsf{SG}}, \mathsf{NIKE.pk}_{\mathsf{SG}}, \mathsf{NIKE.ID}_{\mathsf{R}^*}, \mathsf{NIKE.sk}_{\mathsf{R}^*})$ and $\mathsf{pr}\text{-}kw \leftarrow F_{\mathsf{K}_{\mathsf{NIKE}}}(kw)$. $\mathcal{B}$ sends $\mathsf{pr}\text{-}kw$ to $\mathcal{C}$ as a trapdoor query. $\mathcal{C}$ computes $\mathsf{T}_{\mathsf{PEKS},\mathsf{pr}\text{-}kw} \leftarrow \mathsf{PEKS.Trapdoor}(\mathsf{pp}_{\mathsf{PEKS}}, \mathsf{sk}^*_{\mathsf{PEKS}}, \mathsf{pr}\text{-}kw)$ and returns $\mathsf{T}_{\mathsf{PEKS},\mathsf{pr}\text{-}kw}$ to $\mathcal{B}$. $\mathcal{B}$ returns $\mathsf{T}_{kw} = \mathsf{T}_{\mathsf{PEKS},\mathsf{pr}\text{-}kw}$ to $\mathcal{A}$.

In the challenge phase, $\mathcal{A}$ sends $(\mathsf{ID}_{\mathsf{SG}^*}, kw^*_0, kw^*_1)$ to $\mathcal{B}$. Since $\mathsf{ID}_{\mathsf{SG}^*}$ has been queried to KeyGen and corrupt oracle, $\mathcal{B}$ can retrieve $(\mathsf{NIKE.pk}_{\mathsf{SG}^*}, \mathsf{NIKE.sk}_{\mathsf{SG}^*})$ and runs $\mathsf{K}_{\mathsf{NIKE}} \leftarrow \mathsf{NIKE.SharedKey}(\mathsf{NIKE.ID}_{\mathsf{R}^*}, \mathsf{NIKE.pk}_{\mathsf{R}^*}, \mathsf{NIKE.ID}_{\mathsf{SG}^*}, \mathsf{NIKE.sk}_{\mathsf{SG}^*})$, $\mathsf{pr}\text{-}kw^*_0 \leftarrow F_{\mathsf{K}_{\mathsf{NIKE}}}(kw^*_0)$ and $\mathsf{pr}\text{-}kw^*_1 \leftarrow F_{\mathsf{K}_{\mathsf{NIKE}}}(kw^*_1)$. As mentioned before $\mathcal{B}$ did not send $\mathsf{pr}\text{-}kw^*_0$ and $\mathsf{pr}\text{-}kw^*_1$ as trapdoor queries. $\mathcal{B}$ sends $(\mathsf{pr}\text{-}kw^*_0, \mathsf{pr}\text{-}kw^*_1)$ to $\mathcal{C}$ as the challenge keywords. $\mathcal{C}$ chooses $b \xleftarrow{\$} \{0,1\}$ and returns $\mathsf{C}_{\mathsf{PEKS},\mathsf{pr}\text{-}kw^*_b} \leftarrow \mathsf{PEKS.Enc}(\mathsf{pp}_{\mathsf{PEKS}}, \mathsf{pk}^*_{\mathsf{PEKS}}, \mathsf{pr}\text{-}kw^*_b)$ as the challenge ciphertexts of IND-CKA game. Then $\mathcal{B}$ sends $\mathsf{C}_{\mathsf{PEKS},\mathsf{pr}\text{-}kw^*_b}$ to $\mathcal{A}$ as the challenge cipheretext.

KeyGenSG and Corrupt oracle: The same as before. Note that for $(\mathsf{ID}_{\mathsf{SG}}, kw)$ sent from $\mathcal{A}$, the corresponding $\mathsf{pr}\text{-}kw \notin \{\mathsf{pr}\text{-}kw^*_0, \mathsf{pr}\text{-}kw^*_1\}$. Thus, the simulation is complete.

Trapdoor oracle: The same as before.

$\mathcal{B}$ simulates the insider full CI game and if $\mathcal{A}$ breaks insider full CI, then $\mathcal{B}$ breaks IND-CKA. That is, $\mathsf{Adv}^{\mathsf{InsiderFullCI}}_{\mathsf{PAEKS},\mathcal{A}}(\lambda) \leqq \mathsf{Adv}^{\mathsf{IND\text{-}CKA}}_{\mathsf{PEKS},\mathcal{B}}(\lambda)$ holds. This concludes the proof. □

Theorem 3. *Our PAEKS construction is outsider TI secure if NIKE is HKR-CKS secure, and PRP provides pseudo-randomness.*

Outline of the Security Proof. Ultimately, the value $\mathsf{pr}\text{-}kw^*$ corresponding to the challenge keyword kw^*_b becomes independent of the input, and therefore the adversary's advantage in breaking TI is zero. Prior to this, we need to replace the PRP key with a random value, which can be achieved by the HKR-CKS security of the NIKE scheme. To do so, we need to set the public key of $\mathsf{ID}_{\mathsf{SG}^*}$ to the one generated by the NIKE challenger. For this purpose, we introduce an additional guessing phase to predict when the sender group key generation for $\mathsf{ID}_{\mathsf{SG}^*}$ occurs. Since the number of oracle queries made by the adversary is polynomial in the security parameter, this guess succeeds with non-negligible probability.

Proof. To prove the theorem, we define the sequences of games $\mathsf{Game}_0, \cdots, \mathsf{Game}_3$.

Game_0**:** This game is the same as the original definition of outsider TI13.

Game_1**:** This game is the same as Game_0 except that the following guessing process is introduced. Let q_{SG} be the number of KeyGenSG queries issued by $\mathcal{A}$. $\mathcal{C}$ chooses a random index $i^* \in [q_{SG}]$ at the beginning of the game where $[q_{SG}] = \{1, 2, \ldots, q_{SG}\}$. Let $\mathcal{A}$ send $\mathsf{ID}_{\mathsf{SG}^*}$ to the KeyGenSG oracle as i-th query. If $i \neq i^*$, then the game outputs a random bit and aborts. Obviously,

$$|\Pr[\mathsf{Game}_0 = 1] - 1/2| = (1/q_{SG})|\Pr[\mathsf{Game}_1 = 1] - 1/2|$$

holds.

Game_2**:** This game is the same as Game_1 except that the NIKE key K_{NIKE} used for generating the challenge trapdoor is randomly chosen. Let $\mathcal{C}$ be the challenger of HKR-CKS game. We construct an algorithm $\mathcal{B}_2$ that breaks the HKR-CKS security using $\mathcal{A}$. $\mathcal{C}$ runs $\mathsf{pp}_{\mathsf{NIKE}} \leftarrow \mathsf{NIKE.SetUp}(1^\lambda)$ and sends $\mathsf{pp}_{\mathsf{NIKE}}$ to $\mathcal{B}_2$. $\mathcal{B}_2$ sends $\mathsf{NIKE.ID}_{\mathsf{R}^*}$ to $\mathcal{C}$ as a Register honest user ID query. $\mathcal{C}$ runs $(\mathsf{NIKE.pk}, \mathsf{NIKE.sk}) \leftarrow \mathsf{NIKE.Gen}(\mathsf{NIKE.ID}_{\mathsf{R}^*})$ and sends $\mathsf{NIKE.pk}$ to $\mathcal{B}_2$. $\mathcal{B}_2$ runs $\mathsf{pp}_{\mathsf{PEKS}} \leftarrow \mathsf{PEKS.SetUp}(1^\lambda)$ and $(\mathsf{pk}^*_{\mathsf{PEKS}}, \mathsf{sk}^*_{\mathsf{PEKS}}) \leftarrow \mathsf{PEKS.KGen}(\mathsf{pp}_{\mathsf{PEKS}})$. $\mathcal{B}_2$ sets $\mathsf{pp} = (\mathsf{pp}_{\mathsf{NIKE}}, \mathsf{pp}_{\mathsf{PEKS}})$, $\mathsf{NIKE.pk} = \mathsf{NIKE.pk}_{\mathsf{R}^*}$, and $\mathsf{pk}_{\mathsf{R}^*} = (\mathsf{pk}^*_{\mathsf{PEKS}}, \mathsf{NIKE.pk}_{\mathsf{R}^*})$, and sends $(\mathsf{pp}, \mathsf{pk}_{\mathsf{R}^*})$ to $\mathcal{A}$.

In the following game description, once $\mathcal{B}_2$ knows $\mathsf{NIKE.ID}_{\mathsf{SG}^*}$, $\mathcal{B}_2$ sends $(\mathsf{NIKE.ID}_{\mathsf{SG}^*}, \mathsf{NIKE.ID}_{\mathsf{R}^*})$ as a test query, and reciters $\mathsf{K}^*_{\mathsf{NIKE}}$ that will be used for generating challenge-related ciphertexts and trapdoors.

Phase I:

KeyGenSG Oracle: $\mathcal{A}$ sends $\mathsf{ID}_{\mathsf{SG}}$ where $(\mathsf{ID}_{\mathsf{SG}}, \cdot, \cdot) \notin \mathcal{L}_{\mathsf{KeyGenSG}}$. Due to the modification in Game_1, $\mathcal{B}_2$ knows when $\mathcal{A}$ sends $\mathsf{ID}_{\mathsf{SG}^*}$ to the KeyGenSG oracles. If this is the i^*-th query, i.e., $\mathsf{ID}_{\mathsf{SG}} = \mathsf{ID}_{\mathsf{SG}^*}$ then $\mathcal{B}_2$ sends $\mathsf{ID}_{\mathsf{SG}^*} := \mathsf{NIKE.ID}_{\mathsf{SG}^*}$ to $\mathcal{C}$ as a Register honest user ID query. $\mathcal{C}$ runs $(\mathsf{NIKE.pk}, \mathsf{NIKE.sk}) \leftarrow \mathsf{NIKE.Gen}(\mathsf{NIKE.ID}_{\mathsf{SG}^*})$ and sends $\mathsf{NIKE.pk}$ to $\mathcal{B}_2$. $\mathcal{B}_2$ sets $\mathsf{NIKE.pk} = \mathsf{pk}_{\mathsf{SG}^*}$ and returns $\mathsf{pk}_{\mathsf{SG}^*}$ to $\mathcal{A}$. If this is not is the i^*-th query, $\mathcal{B}_2$ runs $(\mathsf{NIKE.pk}_{\mathsf{SG}}, \mathsf{NIKE.sk}_{\mathsf{SG}}) \leftarrow \mathsf{NIKE.Gen}(\mathsf{NIKE.ID}_{\mathsf{SG}})$, sets $(\mathsf{pk}_{\mathsf{SG}}, \mathsf{sk}_{\mathsf{SG}}) = (\mathsf{NIKE.pk}_{\mathsf{SG}}, \mathsf{NIKE.sk}_{\mathsf{SG}})$, records $\mathcal{L}_{\mathsf{KeyGenSG}} \leftarrow \mathcal{L}_{\mathsf{KeyGenSG}} \cup \{(\mathsf{ID}_{\mathsf{SG}}, \mathsf{pk}_{\mathsf{SG}}, \mathsf{sk}_{\mathsf{SG}})\}$, and returns $\mathsf{pk}_{\mathsf{SG}}$ to $\mathcal{A}$.

CorruptSG Oracle: $\mathcal{A}$ sends $\mathcal{C}$ $\mathsf{ID}_{\mathsf{SG}}$ where $(\mathsf{ID}_{\mathsf{SG}}, \cdot, \cdot) \in \mathcal{L}_{\mathsf{KeyGenSG}}$. $\mathcal{C}$ retrieves $(\mathsf{ID}_{\mathsf{SG}}, \mathsf{pk}_{\mathsf{SG}}, \mathsf{sk}_{\mathsf{SG}}) \in \mathcal{L}_{\mathsf{KeyGenSG}}$ and returns $\mathsf{sk}_{\mathsf{SG}}$ then records the identity $\mathcal{L}_{\mathsf{CorruptSG}} \leftarrow \mathcal{L}_{\mathsf{CorruptSG}} \cup \{\mathsf{ID}_{\mathsf{SG}}\}$. We remark that $\mathcal{A}$ does not send $\mathsf{ID}_{\mathsf{SG}^*}$ and thus $\mathcal{B}_2$ knows all $\mathsf{sk}_{\mathsf{SG}}$.

KeyGenR Oracle: $\mathcal{A}$ sends ID_{R} where $(\mathsf{ID}_{\mathsf{R}}, \cdot, \cdot) \notin \mathcal{L}_{\mathsf{KeyGenR}}$. $\mathcal{B}_2$ sets $\mathsf{ID}_{\mathsf{R}} = \mathsf{NIKE.ID?R}$, runs $(\mathsf{NIKE.pk}_{\mathsf{R}}, \mathsf{NIKE.sk}_{\mathsf{R}}) \leftarrow \mathsf{NIKE.Gen}(\mathsf{NIKE.ID}_{\mathsf{R}})$ and $(\mathsf{pk}_{\mathsf{PEKS}}, \mathsf{sk}_{\mathsf{PEKS}}) \leftarrow \mathsf{PEKS.KGen}(\mathsf{pp}_{\mathsf{PEKS}})$, sets $(\mathsf{pk}_{\mathsf{R}}, \mathsf{sk}_{\mathsf{R}}) = ((\mathsf{pk}_{\mathsf{PEKS}}, \mathsf{NIKE.pk}_{\mathsf{R}}), \mathsf{NIKE.sk}_{\mathsf{R}})$, records $\mathcal{L}_{\mathsf{KeyGenR}} \leftarrow \mathcal{L}_{\mathsf{KeyGenR}} \cup \{(\mathsf{ID}_{\mathsf{R}}, \mathsf{pk}_{\mathsf{R}}, \mathsf{sk}_{\mathsf{R}})\}$, and returns pk_{R} to $\mathcal{A}$.

Encryption Oracle: $\mathcal{A}$ sends $\mathcal{B}_2$ $(\mathsf{ID}_{\mathsf{SG}}, \mathsf{ID}_{\mathsf{R}}, kw)$ where $(\mathsf{ID}_{\mathsf{SG}}, \cdot, \cdot) \in \mathcal{L}_{\mathsf{KeyGenSG}}$, $(\mathsf{ID}_{\mathsf{R}}, \cdot, \cdot) \in \mathcal{L}_{\mathsf{KeyGenR}}$. If $(\mathsf{ID}_{\mathsf{SG}}, \mathsf{ID}_{\mathsf{R}}) = (\mathsf{ID}_{\mathsf{SG}^*}, \mathsf{ID}_{\mathsf{R}^*})$, then $\mathcal{B}_2$ computes $\mathsf{pr}\text{-}kw \leftarrow F_{\mathsf{K}^*_{\mathsf{NIKE}}}(kw)$ and $\mathsf{C}_{\mathsf{PEKS},\mathsf{pr}\text{-}kw} \leftarrow \mathsf{PEKS.Enc}(\mathsf{pp}_{\mathsf{PEKS}}, \mathsf{pk}^*_{\mathsf{PEKS}}, \mathsf{pr}\text{-}kw)$, and returns $\mathsf{C}_{kw} = \mathsf{C}_{\mathsf{PEKS},\mathsf{pr}\text{-}kw}$ to $\mathcal{A}$. If $(\mathsf{ID}_{\mathsf{SG}}, \mathsf{ID}_{\mathsf{R}}) \neq (\mathsf{ID}_{\mathsf{SG}^*}, \mathsf{ID}_{\mathsf{R}^*})$, then $\mathcal{B}_2$ knows either $\mathsf{sk}_{\mathsf{SG}}$ or sk_{R} and thus $\mathcal{B}_2$ can compute either $\mathsf{NIKE.SharedKey}(\mathsf{ID}_{\mathsf{SG}}, \mathsf{sk}_{\mathsf{SG}}, \mathsf{ID}_{\mathsf{R}}, \mathsf{pk}_{\mathsf{R}})$ or $\mathsf{NIKE.SharedKey}(\mathsf{ID}_{\mathsf{SG}}, \mathsf{pk}_{\mathsf{SG}}, \mathsf{ID}_{\mathsf{R}}, \mathsf{sk}_{\mathsf{R}})$. Thus, $\mathcal{B}_2$ can compute a ciphertext.

Trapdoor Oracle: $\mathcal{A}$ sends $\mathcal{B}_2$ $(\mathsf{ID}_{\mathsf{SG}}, kw)$ where $(\mathsf{ID}_{\mathsf{SG}}, \cdot, \cdot) \in \mathcal{L}_{\mathsf{KeyGenSG}}$. If $\mathsf{ID}_{\mathsf{SG}} = \mathsf{ID}_{\mathsf{SG}^*}$, then $\mathcal{B}_2$ computes $\mathsf{pr}\text{-}kw \leftarrow F_{\mathsf{K}^*_{\mathsf{NIKE}}}(kw)$ and $\mathsf{T}_{\mathsf{PEKS},\mathsf{pr}\text{-}kw} \leftarrow \mathsf{PEKS.Trapdoor}(\mathsf{pp}_{\mathsf{PEKS}}, \mathsf{sk}^*_{\mathsf{PEKS}}, \mathsf{pr}\text{-}kw)$, and returns $\mathsf{T}_{kw} = \mathsf{T}_{\mathsf{PEKS},\mathsf{pr}\text{-}kw}$ to $\mathcal{A}$. If $\mathsf{ID}_{\mathsf{SG}} \neq \mathsf{ID}_{\mathsf{SG}^*}$, then

$\mathcal{B}_2$ uses $\mathsf{K}_{\mathsf{NIKE}} \leftarrow \mathsf{NIKE.SharedKey}(\mathsf{ID}_{\mathsf{SG}}, \mathsf{sk}_{\mathsf{SG}}, \mathsf{ID}_{\mathsf{R}^*}, \mathsf{pk}_{\mathsf{R}^*})$ instead of $\mathsf{K}^*_{\mathsf{NIKE}}$. Thus, $\mathcal{B}_2$ can compute a trapdoor.

Challenge Phase: $\mathcal{A}$ declares $(\mathsf{ID}_{\mathsf{SG}^*}, kw_0^*, kw_1^*)$. Note that $\mathcal{B}_2$ has known $\mathsf{ID}_{\mathsf{SG}^*}$ due to the modification in Game_1. $\mathcal{B}_2$ chooses $b \xleftarrow{\$} \{0,1\}$, computes $\mathsf{pr}\text{-}kw^* \leftarrow F_{\mathsf{K}^*_{\mathsf{NIKE}}}(kw_b^*)$ and $\mathsf{T}^*_{\mathsf{PEKS},\mathsf{pr}\text{-}kw} \leftarrow \mathsf{PEKS.Trapdoor}(\mathsf{pp}_{\mathsf{PEKS}}, \mathsf{sk}^*_{\mathsf{PEKS}}, \mathsf{pr}\text{-}kw^*)$, and returns the challenge trapdoor $\mathsf{T}^*_{kw_b^*} = \mathsf{T}_{\mathsf{PEKS},\mathsf{pr}\text{-}kw^*}$ to $\mathcal{A}$.

Phase II: The same as before. We remark that $\mathcal{B}_2$ has responded challenge-related ciphere-texts and trapdoors in Phase I.

If $\mathsf{K}^*_{\mathsf{NIKE}}$ is a NIKE key $\mathsf{SharedKey}(\mathsf{ID}_{\mathsf{SG}^*}, \mathsf{pk}_{\mathsf{SG}^*}, \mathsf{ID}_{\mathsf{R}^*}, \mathsf{NIKE.sk}_{\mathsf{R}^*})$, then $\mathcal{B}_2$ simulates Game_1 and if $\mathsf{K}^*_{\mathsf{NIKE}}$ is random, then $\mathcal{B}_2$ simulates Game_2. Thus,

$$|\Pr[\mathsf{Game}_1 = 1] - \Pr[\mathsf{Game}_2 = 1]| \leq \mathsf{Adv}^{\mathsf{HKR\text{-}CKS}}_{\mathsf{NIKE},\mathcal{B}_2}(\lambda)$$

holds.

Game_3**:** This game is the same as Game_2 except that $\mathsf{pr}\text{-}kw^* \leftarrow F_{\mathsf{K}^*_{\mathsf{NIKE}}}(kw_b^*)$ is changed to $\mathsf{pr}\text{-}kw^* \leftarrow \{0,1\}^{\ell(\lambda)}$. We remark that the simulator in this game does not know (kw_0^*, kw_1^*) in Phase I. However it does not affect simulation due to the restriction of outsider TI where $\mathcal{A}$ is not allowed to send $(\mathsf{ID}_{\mathsf{SG}^*}, \mathsf{ID}_{\mathsf{R}^*}, kw_0^*)$ and $(\mathsf{ID}_{\mathsf{SG}^*}, \mathsf{ID}_{\mathsf{R}^*}, kw_1^*)$ to the encryption oracle, and is not allowed to send $(\mathsf{ID}_{\mathsf{SG}^*}, kw_0^*)$ and $(\mathsf{ID}_{\mathsf{SG}^*}, kw_1^*)$ to the trapdoor oracle. Thus, $\mathsf{pr}\text{-}kw^*$ is computed only once throughout the game. This situation allows us to embed the test result, which is returned by the PRP challenger, to $\mathsf{pr}\text{-}kw^*$. Let $\mathcal{C}$ be the challenger of pseudo randomness, we constructs an algorithm $\mathcal{B}_3$ that breaks pseudo-randomness using $\mathcal{A}$. $\mathcal{B}_2$ runs $\mathsf{pp}_{\mathsf{NIKE}} \leftarrow \mathsf{NIKE.SetUp}(1^\lambda)$, $\mathsf{pp}_{\mathsf{PEKS}} \leftarrow \mathsf{PEKS.SetUp}(1^\lambda)$, $(\mathsf{pk}^*_{\mathsf{PEKS}}, \mathsf{sk}^*_{\mathsf{PEKS}}) \leftarrow \mathsf{PEKS.KGen}(\mathsf{pp}_{\mathsf{PEKS}})$ and $(\mathsf{NIKE.pk}_{\mathsf{R}^*}, \mathsf{NIKE.sk}_{\mathsf{R}^*}) \leftarrow \mathsf{NIKE.Gen}(\mathsf{NIKE.ID}_{\mathsf{R}^*})$. $\mathcal{B}_2$ sets $\mathsf{pp} = (\mathsf{pp}_{\mathsf{NIKE}}, \mathsf{pp}_{\mathsf{PEKS}})$ and $\mathsf{pk}^*_{\mathsf{R}} = (\mathsf{pk}^*_{\mathsf{PEKS}}, \mathsf{NIKE.pk}_{\mathsf{R}^*})$ and sends pp and $\mathsf{pk}^*_{\mathsf{R}}$ to $\mathcal{A}$.

Phase I:

KeyGenSG Oracle: $\mathcal{A}$ sends $\mathcal{B}_3$ $\mathsf{ID}_{\mathsf{SG}}$ where $(\mathsf{ID}_{\mathsf{SG}}, \cdot, \cdot) \notin \mathcal{L}_{\mathsf{KeyGenSG}}$. $\mathcal{B}_3$ runs $(\mathsf{NIKE.pk}_{\mathsf{SG}}, \mathsf{NIKE.sk}_{\mathsf{SG}}) \leftarrow \mathsf{NIKE.Gen}(\mathsf{NIKE.ID})$ and sets $(\mathsf{pk}_{\mathsf{SG}}, \mathsf{sk}_{\mathsf{SG}}) = (\mathsf{NIKE.pk}_{\mathsf{SG}}, \mathsf{NIKE.sk}_{\mathsf{SG}})$ and returns $\mathsf{pk}_{\mathsf{SG}}$ then records $\mathcal{L}_{\mathsf{KeyGenSG}} \leftarrow \mathcal{L}_{\mathsf{KeyGenSG}} \cup \{(\mathsf{ID}_{\mathsf{SG}}, \mathsf{pk}_{\mathsf{SG}}, \mathsf{sk}_{\mathsf{SG}})\}$.

CorruptSG Oracle: $\mathcal{A}$ sends $\mathcal{B}_3$ $\mathsf{ID}_{\mathsf{SG}}$ where $(\mathsf{ID}_{\mathsf{SG}}, \cdot, \cdot) \in \mathcal{L}_{\mathsf{KeyGenSG}}$. $\mathcal{C}$ retrieves $(\mathsf{ID}_{\mathsf{SG}}, \mathsf{pk}_{\mathsf{SG}}, \mathsf{sk}_{\mathsf{SG}}) \in \mathcal{L}_{\mathsf{KeyGenSG}}$ and returns $\mathsf{sk}_{\mathsf{SG}}$ then records the identity $\mathcal{L}_{\mathsf{CorruptSG}} \leftarrow \mathcal{L}_{\mathsf{CorruptSG}} \cup \{\mathsf{ID}_{\mathsf{SG}}\}$.

KeyGenR Oracle: $\mathcal{A}$ sends $\mathcal{B}_3$ ID_{R} where $(\mathsf{ID}_{\mathsf{R}}, \cdot, \cdot) \notin \mathcal{L}_{\mathsf{KeyGenR}}$. $\mathcal{C}$ runs $(\mathsf{pk}_{\mathsf{R}}, \mathsf{sk}_{\mathsf{R}}) \leftarrow \mathsf{KGen}_{\mathsf{R}}(\mathsf{pp}, \mathsf{ID}_{\mathsf{R}})$ and returns pk_{R} then records $\mathcal{L}_{\mathsf{KeyGenR}} \leftarrow \mathcal{L}_{\mathsf{KeyGenR}} \cup \{(\mathsf{ID}_{\mathsf{R}}, \mathsf{pk}_{\mathsf{R}}, \mathsf{sk}_{\mathsf{R}})\}$.

Encryption Oracle: $\mathcal{A}$ sends $\mathcal{B}_3$ $(\mathsf{ID}_{\mathsf{SG}}, \mathsf{ID}_{\mathsf{R}}, kw)$ where $(\mathsf{ID}_{\mathsf{SG}}, \cdot, \cdot) \in \mathcal{L}_{\mathsf{KeyGenSG}}$, $(\mathsf{ID}_{\mathsf{R}}, \cdot, \cdot) \in \mathcal{L}_{\mathsf{KeyGenR}}$. If $(\mathsf{ID}_{\mathsf{SG}}, \mathsf{ID}_{\mathsf{R}}) = (\mathsf{ID}_{\mathsf{SG}^*}, \mathsf{ID}_{\mathsf{R}^*})$, $\mathcal{B}_3$ sends kw to $\mathcal{C}$. $\mathcal{C}$ returns $\mathsf{pr}\text{-}kw$. $\mathcal{B}_3$ computes $\mathsf{C}_{\mathsf{PEKS},\mathsf{pr}\text{-}kw} \leftarrow \mathsf{PEKS.Enc}(\mathsf{pp}_{\mathsf{PEKS}}, \mathsf{pk}^*_{\mathsf{PEKS}}, \mathsf{pr}\text{-}kw)$, and returns $\mathsf{C}_{kw} = \mathsf{C}_{\mathsf{PEKS},\mathsf{pr}\text{-}kw}$ to $\mathcal{A}$. Otherwise, if $(\mathsf{ID}_{\mathsf{SG}}, \mathsf{ID}_{\mathsf{R}}) \neq (\mathsf{ID}_{\mathsf{SG}^*}, \mathsf{ID}_{\mathsf{R}^*})$, $\mathcal{B}_3$ retrieves $(\mathsf{ID}_{\mathsf{SG}}, \mathsf{pk}_{\mathsf{SG}}, \mathsf{sk}_{\mathsf{SG}}) \in \mathcal{L}_{\mathsf{KeyGenSG}}$ and $(\mathsf{ID}_{\mathsf{R}}, \mathsf{pk}_{\mathsf{R}}, \mathsf{sk}_{\mathsf{R}}) \in \mathcal{L}_{\mathsf{KeyGenR}}$ and runs $\mathsf{pr}\text{-}kw \leftarrow F_{K_{\mathsf{NIKE}}}(kw)$ and $\mathsf{C}_{\mathsf{PEKS},\mathsf{pr}\text{-}kw} \leftarrow \mathsf{PEKS.Enc}(\mathsf{pp}_{\mathsf{PEKS}}, \mathsf{pk}^*_{\mathsf{PEKS}}, \mathsf{pr}\text{-}kw)$, and returns $\mathsf{C}_{kw} = \mathsf{C}_{\mathsf{PEKS},\mathsf{pr}\text{-}kw}$ to $\mathcal{A}$.

Trapdoor Oracle: $\mathcal{A}$ sends $\mathcal{B}_3$ $(\mathsf{ID}_{\mathsf{SG}}, kw)$ where $(\mathsf{ID}_{\mathsf{SG}}, \cdot, \cdot) \in \mathcal{L}_{\mathsf{KeyGenSG}}$. If $\mathsf{ID}_{\mathsf{SG}} = \mathsf{ID}_{\mathsf{SG}^*}$, $\mathcal{B}_3$ sends kw to $\mathcal{C}$. $\mathcal{C}$ returns $\mathsf{pr}\text{-}kw$. $\mathcal{B}_3$ computes $\mathsf{T}_{\mathsf{PEKS},\mathsf{pr}\text{-}kw} \leftarrow \mathsf{PEKS.Trapdoor}$

$(\mathsf{pp}_{\mathsf{PEKS}}, \mathsf{sk}^*_{\mathsf{PEKS}}, \mathsf{pr}\text{-}kw)$ then returns $\mathsf{T}_{kw} = \mathsf{T}_{\mathsf{PEKS},\mathsf{pr}\text{-}kw}$ to $\mathcal{A}$. Otherwise, if $\mathsf{ID}_{\mathsf{SG}} \neq \mathsf{ID}_{\mathsf{SG}^*}$, $\mathcal{B}_3$ retrieve $(\mathsf{ID}_{\mathsf{SG}}, \mathsf{pk}_{\mathsf{SG}}, \mathsf{sk}_{\mathsf{SG}}) \in \mathcal{L}_{\mathsf{KeyGenSG}}$ and runs $\mathsf{pr}\text{-}kw \leftarrow F_{K_{\mathsf{NIKE}}}(kw)$ and $\mathsf{T}_{\mathsf{PEKS},\mathsf{pr}\text{-}kw} \leftarrow \mathsf{PEKS.Trapdoor}(\mathsf{pp}_{\mathsf{PEKS}}, \mathsf{sk}^*_{\mathsf{PEKS}}, \mathsf{pr}\text{-}kw)$ then returns $\mathsf{T}_{kw} = \mathsf{T}_{\mathsf{PEKS},\mathsf{pr}\text{-}kw}$ to $\mathcal{A}$.

Challenge phase: $\mathcal{A}$ declares $(\mathsf{ID}_{\mathsf{SG}^*}, kw^*_0, kw^*_1)$. $\mathcal{B}_2$ chooses $b \xleftarrow{\$} \{0,1\}$ and sends kw^*_b to $\mathcal{C}$. $\mathcal{C}$ returns $\mathsf{pr}\text{-}kw^*$. $\mathcal{B}_2$ computes $\mathsf{T}^*_{\mathsf{PEKS},prkw} \leftarrow \mathsf{PEKS.Trapdoor}(\mathsf{pp}_{\mathsf{PEKS}}, \mathsf{sk}^*_{\mathsf{PEKS}}, \mathsf{pr}\text{-}kw^*)$, and returns the challenge trapdoor $\mathsf{T}^*_{kw^*_b} = \mathsf{T}_{\mathsf{PEKS},\mathsf{pr}\text{-}kw^*}$ to $\mathcal{A}$.

Phase II: The same as before. We remark that $\mathcal{B}_3$ has responded challenge-related cipheretexts and trapdoors in Phase I.

If $\mathsf{pr}\text{-}kw^*$ associated to $\mathsf{T}_{\mathsf{PEKS},\mathsf{pr}\text{-}kw^*}$ is uniformly random bit string, $\mathcal{B}_3$ simulates Game_3 and $\mathcal{B}_3$ simulates Game_2, otherwise. Thus,

$$|\Pr[\mathsf{Game}_2 = 1] - \Pr[\mathsf{Game}_3 = 1]| \leq \mathsf{Adv}^{\mathsf{Pseudo\text{-}random}}_{\mathsf{PRP},\mathcal{B}_3}(\lambda)$$

holds.

In Game_3, $|\Pr[\mathsf{Game}_3 = 1] - 1/2| = 0$ holds since $\mathsf{pr}\text{-}kw^*$ is independent of b. Putting it altogether, we have

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{OutsiderTI}}_{\mathsf{PAEKS},\mathcal{A}}(\lambda) &= |\Pr[\mathsf{Game}_0 = 1] - 1/2| \\
&= (1/q_{SG})|\Pr[\mathsf{Game}_1 = 1] - 1/2| \\
&\leqq (1/q_{SG})((\sum_{i=1}^{2} |\Pr[\mathsf{Game}_i = 1] - \Pr[\mathsf{Game}_{(i+1)} = 1]|) \\
&\quad + |\Pr[\mathsf{Game}_3 = 1] - 1/2|) \\
&\leqq (1/q_{SG})(\mathsf{Adv}^{\mathsf{HKR\text{-}CKS}}_{\mathsf{NIKE},\mathcal{B}_2}(\lambda) + \mathsf{Adv}^{\mathsf{Pseudo\text{-}random}}_{\mathsf{PRP},\mathcal{B}_3}(\lambda))
\end{aligned}$$

This concludes the proof. □

6 Conclusion

In this paper, we proposed a generic construction of PAEKS based on NIKE, PRP, and PEKS, where the search complexity depends on the number of sender groups. The proposed generic construction can be instantiated either pairing or lattices. To ensure that no keyword information is revealed from ciphertexts, even if they are generated by other members of the same sender group, we introduced insider full CI security. Moreover, due to the impossible result in [35], we introduced outsider TI security. Providing outsider full TI remains an interesting direction for future work.

To the best of our knowledge, for the other construction methodologies introduced in Section 1, transformation-oriented [24, 25, 28, 32, 38] and server-oriented [11, 13, 33] approaches, only concrete instantiations have been provided so far, and no generic constructions are known. Providing such generic constructions is useful for understanding the feasibility of these approaches. Furthermore, by employing provably secure building blocks in generic constructions, it is possible to avoid construction-specific attacks such as those discussed in this paper [17,18,36,37]. Extending our approach to develop generic constructions for other methodologies is also left as an important direction for future work.

Acknowledgment: This work was supported by JSPS KAKENHI Grant Number JP26K02908.

References

[1] Michel Abdalla, Mihir Bellare, Dario Catalano, Eike Kiltz, Tadayoshi Kohno, Tanja Lange, John Malone-Lee, Gregory Neven, Pascal Paillier, and Haixia Shi. Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. *Journal of Cryptology*, 21(3):350–391, 2008.

[2] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Modular design of secure group messaging protocols and the security of MLS. In *ACM CCS*, pages 1463–1483. ACM, 2021.

[3] Daniel Apon, Dana Dachman-Soled, Huijing Gong, and Jonathan Katz. Constant-round group key exchange from the ring-LWE assumption. In *Post-Quantum Cryptography*, pages 189–205, 2019.

[4] David Balbás, Daniel Collins, and Serge Vaudenay. Cryptographic administration for secure group messaging. In *USENIX Security Symposium*, pages 1253–1270. USENIX Association, 2023.

[5] Fabrice Benhamouda, Olivier Blazy, Céline Chevalier, David Pointcheval, and Damien Vergnaud. New techniques for sphfs and efficient one-round PAKE protocols. In *CRYPTO, Part I*, pages 449–475, 2013.

[6] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *EUROCRYPT*, pages 506–522, 2004.

[7] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *CRYPTO*, pages 213–229, 2001.

[8] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *ASIACRYPT*, pages 514–532, 2001.

[9] Xavier Boyen and Brent Waters. Anonymous hierarchical identity-based encryption (without random oracles). In *CRYPTO*, pages 290–307, 2006.

[10] Mike Burmester and Yvo Desmedt. A secure and efficient conference key distribution system (extended abstract). In *EUROCRYPT*, pages 275–286, 1994.

[11] Jiayin Cai, Xingwen Zhao, Dexin Li, Hui Li, and Kai Fan. Efficient and expressive public key authenticated encryption with keyword search in multi-user scenarios. *CoRR*, abs/2503.16828, 2025.

[12] David Cash, Eike Kiltz, and Victor Shoup. The twin Diffie–Hellman problem and applications. *Journal of Cryptology*, 22(4):470–504, 2009.

[13] Leixiao Cheng and Fei Meng. Server-aided public key authenticated searchable encryption with constant ciphertext and constant trapdoor. *IEEE Transactions on Information Forensics and Security*, 19:1388–1400, 2024.

[14] Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican, and Kevin Milner. On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. In *ACM CCS*, pages 1802–1819, 2018.

[15] Cas Cremers, Britta Hale, and Konrad Kohbrok. The complexities of healing in secure group messaging: Why cross-group effects matter. In *USENIX Security Symposium*, pages 1847–1864. USENIX Association, 2021.

[16] Keita Emura. Generic construction of public-key authenticated encryption with keyword search revisited: Stronger security and efficient construction. In *ACM ASIA Public-Key Cryptography Workshop*, pages 39–49, 2022.

[17] Keita Emura. Comments on "Lightweight multi-user public-key authenticated encryption with keyword search". *IEEE Transactions on Information Forensics and Security*, 21:5108–5109, 2026.

[18] Keita Emura. Security analysis on the multi-user keyword-field-free conjunctive searchable encryption scheme. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 2026, to appear. `https://doi:10.1587/10.1587/transfun.2025EAL2101`.

[19] Eduarda S. V. Freire, Dennis Hofheinz, Eike Kiltz, and Kenneth G. Paterson. Non-interactive key exchange. In *Public-Key Cryptography*, pages 254–271, 2013.

[20] Phillip Gajland, Bor de Kock, Miguel Quaresma, Giulio Malavolta, and Peter Schwabe. SWOOSH: efficient lattice-based non-interactive key exchange. In *USENIX Security*, 2024.

[21] Qiong Huang and Hongbo Li. An efficient public-key searchable encryption scheme secure against inside keyword guessing attacks. *Information Sciences*, 403:1–14, 2017.

[22] David Inyangson, Sarah Radway, Tushar M. Jois, Nelly Fazio, and James Mickens. Amigo: Secure group mesh messaging in realistic protest settings. In *ACM CCS*, pages 4244–4258. ACM, 2025.

[23] Jonathan Katz and Vinod Vaikuntanathan. Round-optimal password-based authenticated key exchange. In *Theory of Cryptography*, pages 293–310, 2011.

[24] Hongbo Li, Qiong Huang, Jianye Huang, and Willy Susilo. Public-key authenticated encryption with keyword search supporting constant trapdoor generation and fast search. *IEEE Transactions on Information Forensics and Security*, 18:396–410, 2023.

[25] Hongbo Li, Willy Susilo, Jian Shen, Chen Wang, Leixiao Cheng, and Qiong Huang. Proxy-free public-key authenticated updatable and searchable encryption for cloud storage. *IEEE Transactions on Dependable and Secure Computing*, 23(1):236–249, 2026.

[26] Qinyi Li and Xavier Boyen. Public-key authenticated encryption with keyword search made easy. *IACR Communications in Cryptology*, 1(2):16, 2024.

[27] Zi-Yuan Liu, Yi-Fan Tseng, Raylin Tso, Masahiro Mambo, and Yu-Chi Chen. Public-key authenticated encryption with keyword search: Cryptanalysis, enhanced security, and quantum-resistant instantiation. In *ASIA CCS*, pages 423–436. ACM, 2022.

[28] Fucai Luo, Haiyan Wang, and Xingfu Yan. Re-PAEKS: Public-key authenticated re-encryption with keyword search. *IEEE Transactions on Mobile Computing*, 23(10):10077–10092, 2024.

[29] Jae Hong Seo, Tetsutaro Kobayashi, Miyako Ohkubo, and Koutarou Suzuki. Anonymous hierarchical identity-based encryption with constant size ciphertexts. In *Public Key Cryptography*, pages 215–234, 2009.

[30] Théophile Wallez, Jonathan Protzenko, and Karthikeyan Bhargavan. TreeKEM: A modular machine-checked symbolic security analysis of group key agreement in messaging layer security. In *IEEE Symposium on Security and Privacy*, pages 4375–4390. IEEE, 2025.

[31] Wei Wang, Dongli Liu, Zilin Zheng, Peng Xu, and Laurence Tianruo Yang. Identity-based group encryption with keyword search against keyword guessing attack. *IEEE Transactions on Information Forensics and Security*, 19:8023–8036, 2024.

[32] Yongliang Xu, Hang Cheng, Jiguo Li, Ximeng Liu, Xinpeng Zhang, and Meiqing Wang. Lightweight multi-user public-key authenticated encryption with keyword search. *IEEE Transactions on Information Forensics and Security*, 20:3234–3246, 2025.

[33] Wenjie Yang, Shujie Lin, Futai Zhang, and Anjia Yang. Dual-server public key authenticated searchable encryption with constant trapdoor against inside keyword guessing attacks. *IEEE Transactions on Information Forensics and Security*, pages 1–1, 2026 (Early Access).

[34] Takeshi Yoshida and Keita Emura. On the relations between matchmaking public key encryption and public key authenticated encryption with keyword search. In *Cryptology and Network Security*, pages 272–291, 2025.

[35] Takeshi Yoshida and Keita Emura. Comments on "Identity-based group encryption with keyword search against keyword guessing attack". *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 109(7):1314–1315, 2026.

[36] Takeshi Yoshida and Keita Emura. Comments on "Server-aided public key authenticated searchable encryption with constant ciphertext and constant trapdoor". Cryptology ePrint Archive, Paper 2026/1015, 2026.

[37] Takeshi Yoshida and Keita Emura. On the security of public key authenticated encryption with keyword search with sender-independent search complexity. Cryptology ePrint Archive, Paper 2026/1019, 2026.

[38] Shengjie Zhang, Jianchang Lai, Liquan Chen, Jinguang Han, and Ge Wu. Keyword-field-free conjunctive searchable encryption for multiuser in EMR system. *IEEE Internet of Things Journal*, 12(15):30850–30861, 2025.