A Generic Construction of Tightly Secure Password-Based Authenticated Key Exchange

Jiaxin Pan[1,2(✉)] and Runzhi Zeng[2]

[1] University of Kassel, Kassel, Germany
[2] Department of Mathematical Sciences, NTNU - Norwegian University of Science and Technology, Trondheim, Norway
{jiaxin.pan,runzhi.zeng}@ntnu.no

Abstract. We propose a generic construction of password-based authenticated key exchange (PAKE) from key encapsulation mechanisms (KEM). Assuming that the KEM is oneway secure against plaintext-checkable attacks (OW-PCA), we prove that our PAKE protocol is *tightly secure* in the Bellare-Pointcheval-Rogaway model (EUROCRYPT 2000). Our tight security proofs require ideal ciphers and random oracles. The OW-PCA security is relatively weak and can be implemented tightly with the Diffie-Hellman assumption, which generalizes the work of Liu et al. (PKC 2023), and "almost" tightly with lattice-based assumptions, which tightens the security loss of the work of Beguinet et al. (ACNS 2023) and allows more efficient practical implementation with Kyber. Beyond these, it opens an opportunity of constructing tight PAKE based on various assumptions.

Keywords: Password-based authenticated key exchange · generic constructions · tight security · lattices

1 Introduction

While authenticated key exchange (AKE) protocols require a PKI to certify user public keys, password-based AKE (PAKE) protocols allow a client and a server to establish a session key, assuming that both parties share a password in advance. A password is chosen from a small set of possible strings, referred as a dictionary. Thus, a password has low-entropy and can be memorized by humans. Hence, it is very convenient, and the design and analysis of PAKE protocols have drew a lot of attention in the past few years.

After the introduction of Encrypted-Key-Exchange (EKE) protocol by Bellovin and Merritt [12], many PAKE protocols have been proposed based on variants of the Diffie-Hellman assumptions, including the well-known SPEKE [22], SPEKE2 [6], J-PAKE [20], and CPace [19]. There are only a few exception

Supported by the Research Council of Norway under Project No. 324235.

J. Guo and R. Steinfeld (Eds.): ASIACRYPT 2023, LNCS 14445, pp. 143–175, 2023.
https://doi.org/10.1007/978-981-99-8742-9_5

where PAKE is constructed based on *post-quantum assumptions*, such as lattices [13, 23, 33] and group actions [4].

<u>SECURITY OF PAKE.</u> The security requirements on a PAKE protocol are resistance against offline (where an adversary performs an exhaustive search for the password offline) and online (where an active adversary tries a small number of passwords to run the protocol) dictionary attacks. Similar to the classical AKE, forward secrecy is required as well, where the session keys remain secure, even if the password is corrupted at a later point in time, and also leakage of a session key should not affect other session keys. Their security is formalized by either the indistinguishability-based (IND-based) model [10] or the universal composability (UC) framework [16].

Usually, the advantage of a PAKE protocol $\varepsilon_{\text{PAKE}}$ has the form of:

$$\varepsilon_{\text{PAKE}} \leq {}^{S}/_{|\mathcal{PW}|} + L \cdot \varepsilon_{\text{Problem}}, \tag{1}$$

where S is the number of protocol sessions, $\mathcal{PW}$ is the set of all possible passwords, $\varepsilon_{\text{Problem}}$ is the advantage of attacking the underlying cryptographic hard problem, and L is called the security loss. Here we ignore the additive statistical negligible probability in Eq. (1) for simplicity. Essentially, ${}^{S}/_{|\mathcal{PW}|}$ is the success probability of online dictionary attacks and Eq. (1) shows that the best attack on the PAKE protocol is performing an online dictionary attack. This can be eliminated by restricting the online password guess in practice.

<u>TIGHT SECURITY.</u> We say a security proof for PAKE tight if L is a small constant. All the aforementioned PAKE protocols are non-tight. For instance, according to the analysis of [8], we estimate that the security loss L for the EKE protocol is $O(q_D \cdot (S + q_D))$, where q_D is the number of the adversary's queries to an ideal cipher. The security bound for the group-action-based protocol $\mathsf{Com\text{-}GA\text{-}PAKE}_\ell$ in [4] is even worse, and it contains a square root of the advantage of the underlying assumption (cf. [4, Theorem 2]), due to the Reset Lemma [9]. This means even if we set up the underlying assumption with 128-bit security, $\mathsf{Com\text{-}GA\text{-}PAKE}_\ell$ in [4] has only less[1] than 64-bit.

We note that $\mathsf{X\text{-}GA\text{-}PAKE}_\ell$ in [4, Section 6] has tight security by restricting to weak forward secrecy, where an adversary is not allowed to perform active attacks before password corruptions. This is a rather weak security model.

In this paper, we are interested in tightly secure PAKE with perfect forward secrecy (PFS), namely, adversaries can perform active attacks before password corruptions. From a theoretical perspective, it is interesting to analyze the possibility of constructing tightly secure PAKE and under which cryptographic assumption it is possible. From a practical perspective, it is very desirable to have tightly secure PAKE (or AKE in general), since these protocols are executed in a multi-user, multi-instance scenario. In today's internet, the scenario size is often large. A non-tight protocol requires a larger security parameter to compensate the security loss and results in a less efficient protocol. Even if we

[1] This is because of the additional multiplicative loss factor depending on S and the length of a password in [4, Theorem 2].

cannot achieve full tightness, a tighter security proof is already more beneficial than a less tight one of the same protocol, since the tighter proof offers higher security guarantees.

Our Goal: Tight PAKE Beyond Diffie-Hellman (DH). There are a few exceptions that construct tight PAKE protocols with PFS, and they are all based on the DH assumption. Becerra et al. [7] proved tight security of the three-move PAK protocol [25] using the Gap DH (GDH) assumption [26] in the IND-based model, where the GDH assumption states that the Computational DH (CDH) assumption is hard even if the Decisional DH (DDH) assumption is easy. Lately, Abdalla et al. [2] proved tight security of two-move SPAKE2 in the relaxed UC framework under the GDH assumption. Very recently, Liu et al. [24] carefully used the twinning technique [17] to remove the GDH assumption and proved a variant of the EKE protocol tightly based on the CDH assumption.

Our goal is to construct tightly secure PAKE protocols from post-quantum assumptions, beyond the DH assumptions. Lattice-based assumptions are the promising post-quantum ones, and it seems inherent that they do not have any Gap-like assumption or twinning techniques, since the Decisional and Computational variants of, for instance, Learning-With-Errors (LWE) assumption [30] are equivalent.

Regarding the assumption based on group actions, as we discussed earlier, the $\mathsf{Com\text{-}GA\text{-}PAKE}_\ell$ protocol in [4] needs to rewind an adversary to argue PFS, and by using the Reset Lemma it leads to a very loose bound. Apart from that, $\mathsf{Com\text{-}GA\text{-}PAKE}_\ell$ applies the group action in a "bit-by-bit" (wrt the bit-length of a password) fashion and sends out the resulting element, and thus it is quite inefficient in terms of both computation and communication complexity.

Finally, we note that Liu et al. [24] did not provide a formal proof on the PFS of their protocol, but rather an informal remark. In [4], we note a huge gap between the security loss of a weak FS protocol and a PFS one. Hence, in this paper we will prove the PFS of our protocol concretely.

1.1 Our Contribution

We propose a generic construction of tightly secure PAKE protocols from key encapsulation mechanisms (KEMs) in the ideal cipher and random oracle models. We require the underlying KEM to have the following security:

- Oneway plaintext-checking (OW-PCA) security in the multi-user, multi-challenge setting, namely, adversary $\mathcal{A}$'s goal is to decapsulate one ciphertext out of many given ones, and furthermore, $\mathcal{A}$ is given an oracle to check whether a key k is a valid decapsulation of a ciphertext c under some user j. It is a (slight) multi-user, multi-challenge variant of the original OW-PCA [27].
- Anonymous ciphertexts under PCA, namely, the challenge ciphertexts do not leak any information about the corresponding public keys.
- Fuzzy public keys, namely, the generated public keys are indistinguishable from a random key from all the possible public keys.

Such a KEM can be tightly constructed:

– either *generically* from pseudorandom PKE against chosen-plaintext attacks in the multi-user, multi-challenge setting (PR-CPA security[2]), which states that the given challenge ciphertexts are pseudorandom. This means, as long as we have a PR-CPA secure PKE, we have a PAKE protocol that preserves the tightness of the PKE. With lattices, we do not know a tightly PR-CPA PKE, but only a scheme (i.e. Regev's encryption [30]) tightly wrt. the number of challenges, not wrt. the number of users. This already results in a tighter PAKE protocol than the analysis from Beguinet et al. [8]. More details will be provided in "COMPARISON USING KYBER".
– or *directly* from the strong DH (stDH) assumption in a prime-order group [3]. Under this stronger assumption, our resulting PAKE protocol has $O(\lambda)$ (which corresponds to the bit-length of a group element) less than the 2DH-EKE protocol of Liu et al. [24] in terms of protocol transcripts. In fact, using the twinning technique of Cash et al. [17], we can remove the strong oracle and have our protocol under the CDH assumption, which is the same protocol as the 2DH-EKE protocol of Liu et al. Essentially, our direct instantiation abstracts the key ideas of Liu et al., and our proof for PFS gives a formal analysis of Liu et al.'s protocol.

Different to other PAKE protocol from group actions [4] and lattices as in [13], our construction is compact and does not use "bit-by-bit" approaches. Figure 1 briefly summarizes our approaches.

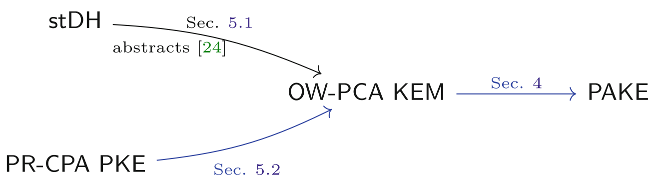

Fig. 1. Overview of our construction. All implications are tight, and the blue ones are done via generic constructions. OW-PCA security is the core for our "KEM-to-PAKE" transformation. Please find additional requirements on the KEM in the text. (Color figure online)

Our proofs are in the IND-based model (aka, the so-called Bellare-Pointcheval-Rogaway (BPR) model [10]) for readability. We are optimistic that it is tightly secure in the UC framework and briefly sketch the ideas about how to lift our proofs in the BPR model to the UC framework in our full version [28].

COMPARISON USING KYBER [32]. There are only a few efficient PAKE protocols from lattices. We focus our comparison on the very efficient one by implementing the CAKE in [8] with KYBER. The reason of not using OCAKE in [8] is because

[2] Our security notions are in the multi-user, multi-challenge setting. Hence, for simplicity, we do not write the 'm' in the abbreviations.

OCAKE do not have PFS, but weak FS. Our protocol is similar to CAKE, but ours has tight reductions from the KEM security.

Unfortunately, by implementing with Kyber, our protocol does not have tight security, since we cannot prove tight PR-CPA security for Kyber, but in practice one will consider using Kyber than otherwise. Our security loss is $O(S \cdot (S + q_D))$ to the Module-LWE assumption, while the security loss of CAKE is $O(q_D \cdot (S + q_D))$, where q_D is the number of decryption queries to the ideal cipher. In practice, q_D is the number of adversary $\mathcal{A}$ evaluating the symmetric cipher offline and can be large. We assume $q_D = 2^{40}$.

Very different to the standard AKE, in the PAKE setting S should be very small, since S corresponds to how many attempts an adversary can perform online dictionary attacks. We usually will limit it. We assume $S \leq 100 \approx 2^6$. Hence, although our security bound with Kyber is not tight, it is still much smaller than CAKE, since $S \ll q_D$. In fact, we have doubt on the security proof of CAKE in handling reply attacks[3], namely, $\mathcal{A}$ can reply the first round message. To fix it, we need to introduce another multiplicative factor S, but since S is relatively small we ignore it in our comparison.

Hence, implementing with Kyber-768 (corresponding to AES-192), our protocol provides about 152-bit security, while CAKE about 112-bit security.

Open Problem. We are optimistic that our protocol can be proven tightly in the weaker and more efficient randomized half-ideal cipher model [31], and we leave the formal proof for it as an open problem.

2 Preliminaries

For an integer n, we define the notation $[n] := \{1, \ldots, n\}$. Let $\mathcal{X}$ and $\mathcal{Y}$ be two finite sets. The notation $x \xleftarrow{\$} \mathcal{X}$ denotes sampling an element x from $\mathcal{X}$ uniformly at random.

Let $\mathcal{A}$ be an algorithm. If $\mathcal{A}$ is probabilistic, then $y \leftarrow \mathcal{A}(x)$ means that the variable y is assigned to the output of $\mathcal{A}$ on input x. If $\mathcal{A}$ is deterministic, then we may write $y := \mathcal{A}(x)$. We write $\mathcal{A}^{\mathcal{O}}$ to indicate that $\mathcal{A}$ has classical access to oracle $\mathcal{O}$, and $\mathcal{A}^{|\mathcal{O}\rangle}$ to indicate that $\mathcal{A}$ has quantum access to oracle $\mathcal{O}$ All algorithms in this paper are probabilistic polynomial-time (PPT), unless we mention it.

Games. We use code-based games [11] to define and prove security. We implicitly assume that Boolean flags are initialized to false, numerical types are initialized to 0, sets and ordered lists are initialized to $\emptyset$, and strings are initialized to the empty string ϵ. The notation $\Pr[\mathbf{G}^{\mathcal{A}} \Rightarrow 1]$ denotes the probability that the final output $\mathbf{G}^{\mathcal{A}}$ of game $\mathbf{G}$ running an adversary $\mathcal{A}$ is 1. Let $\mathtt{Ev}$ be an (classical) event. We write $\Pr[\mathtt{Ev} : \mathbf{G}]$ to denote the probability that $\mathtt{Ev}$ occurs during the game $\mathbf{G}$. In our security notions throughout the paper, we let N, μ be numbers

[3] More precisely, the argument in [8, page 41] under "*Analysis*" may not hold true for reply attacks.

of users and challenges, respectively, which are assumed to be polynomial in the security parameter λ. For simplicity, in this paper, we do not write λ explicitly. Instead, we assume every algorithm's input includes λ.

2.1 Key Encapsulation Mechanism

Definition 1 (Key Encapsulation Mechanism). *A KEM* KEM *consists of four algorithms* $(\mathsf{Setup}, \mathsf{KG}, \mathsf{Encaps}, \mathsf{Decaps})$ *and a ciphertext space* $\mathcal{C}$*, a randomness space* $\mathcal{R}$*, and a KEM key space* $\mathcal{K}$*. On input security parameters,* Setup *outputs a system parameter* par*.* $\mathsf{KG}(\mathsf{par})$ *outputs a public and secret key pair* $(\mathsf{pk}, \mathsf{sk})$*. The encapsulation algorithm* Encaps*, on input* pk*, outputs a ciphertext* $\mathsf{c} \in \mathcal{C}$*. We also write* $\mathsf{c} := \mathsf{Encaps}(\mathsf{pk}; r)$ *to indicate the randomness* $r \in \mathcal{R}$ *explicitly. The decapsulation algorithm* Decaps*, on input* sk *and a ciphertext* c*, outputs a KEM key* $\mathsf{k} \in \mathcal{K}$ *or a rejection symbol* $\perp \notin \mathcal{K}$*. Here* Encaps *and* Decaps *also take* par *as input, but for simplicity, we do not write explicitly.*

Definition 2 (KEM Correctness). *Let* $\mathsf{KEM} := (\mathsf{Setup}, \mathsf{KG}, \mathsf{Encaps}, \mathsf{Decaps})$ *be a KEM scheme and* $\mathcal{A}$ *be an adversary against* KEM*. We say* KEM *is* $(1-\delta)$*-correct if*

$$\Pr\left[(\mathsf{c}, \mathsf{k}) \leftarrow \mathsf{Encaps}(\mathsf{pk}) \wedge \mathsf{k} \neq \mathsf{Decaps}(\mathsf{sk}, \mathsf{c})\right] \leq \delta,$$

where $\mathsf{par} \leftarrow \mathsf{Setup}, (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}(\mathsf{par})$*.*

Definition 3 (Implicit Rejection [14]). *A KEM scheme* $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{KG}, \mathsf{Encaps}, \mathsf{Decaps})$ *has implicit rejection if* $\mathsf{Decaps}(\mathsf{sk}, \cdot)$ *behaves as a pseudorandom function when the input ciphertext is invalid, where* $\mathsf{par} \leftarrow \mathsf{Setup}, (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}$*, and* sk *is the key of the pseudorandom function. That is, if an input ciphertext* c *is invalid, then* $\mathsf{Decaps}(\mathsf{sk}, \mathsf{c})$ *will output a pseudorandom key* k *instead of a rejection symbol* $\perp$*. A concrete example is shown in Fig. 18.*

$\mathsf{OW\text{-}PCA}$ Security. Let $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{KG}, \mathsf{Encaps}, \mathsf{Decaps})$ be a KEM scheme with ciphertext space $\mathcal{C}$. In Definitions 4 and 5, we define two variants of one-wayness under plaintext-checking attacks ($\mathsf{OW\text{-}PCA}$) security for KEM [27] in the multi-user, multi-challenge setting. They will be used for the tight security proof of our PAKE protocol and can be instantiated tightly from the Diffie-Hellman assumption and Learning-With-Errors assumption. Instead of writing 'm' in the abbreviation, we mention the explicit numbers of users and challenge ciphertexts as N and μ in the abbreviation of security.

Definition 4 (Multi-user-challenge OW-PCA security). *Let* N *and* μ *be the numbers of users and challenge ciphertexts per user, respectively. Let* $\mathcal{A}$ *be an adversary against* KEM*. We define the* (N, μ)*-OW-PCA advantage function of* $\mathcal{A}$ *against* KEM

$$\mathsf{Adv}_{\mathsf{KEM}}^{(N,\mu)\text{-}\mathsf{OW\text{-}PCA}}(\mathcal{A}) := \Pr\left[\mathsf{OW\text{-}PCA}_{\mathsf{KEM}}^{(N,\mu),\mathcal{A}} \Rightarrow 1\right],$$

where the game $\mathsf{OW\text{-}PCA}_{\mathsf{KEM}}^{(N,\mu),\mathcal{A}}$ *is defined in Fig. 2. We say* KEM *is OW-PCA secure if* $\mathsf{Adv}_{\mathsf{KEM}}^{(N,\mu)\text{-}\mathsf{OW\text{-}PCA}}(\mathcal{A})$ *is negligible for any* $\mathcal{A}$*.*

GAME $\mathsf{OW\text{-}PCA}_{\mathsf{KEM}}^{(N,\mu),\mathcal{A}}$

$\mathsf{par} \leftarrow \mathsf{Setup}$
for $i \in [N]$
  $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}(\mathsf{par})$
  $(\mathbf{pk}[i], \mathbf{sk}[i]) := (\mathsf{pk}, \mathsf{sk})$
  for $j \in [\mu]$:
    $(\mathsf{c}, \mathsf{k}) \leftarrow \mathsf{Encaps}(\mathbf{pk}[i])$
    $(\mathbf{c}[i,j], \mathbf{k}[i,j]) := (\mathsf{c}, \mathsf{k})$
$(i, j, \mathsf{k}^*) \leftarrow \mathcal{A}^{\mathrm{PCO}}(\mathsf{pk}, \mathbf{c})$
return $\mathsf{k}^* == \mathsf{Decaps}(\mathbf{sk}[i], \mathbf{c}[i,j])$

GAME $\mathsf{OW\text{-}rPCA}_{\mathsf{KEM}}^{(N,\mu),\mathcal{A}}$

$\mathsf{par} \leftarrow \mathsf{Setup}$
for $i \in [N]$
  $(\mathbf{pk}[i], \mathbf{sk}[i]) := (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}(\mathsf{par})$
for $j \in [N \cdot \mu]$:
  $\mathbf{c}[j] := \mathsf{c} \xleftarrow{\$} \mathcal{C}$
$(i, j, \mathsf{k}^*) \leftarrow \mathcal{A}^{\mathrm{PCO}}(\mathsf{pk}, \mathbf{c})$
return $\mathsf{k}^* == \mathsf{Decaps}(\mathbf{sk}[i], \mathbf{c}[j])$

Oracle $\mathrm{PCO}(i, \mathsf{c}, \mathsf{k})$

if $\mathbf{pk}[i] = \perp$
  return $\perp$
return $\mathsf{k} == \mathsf{Decaps}(\mathbf{sk}[i], \mathsf{c})$

Fig. 2. Security games OW-PCA and OW-rPCA for KEM scheme KEM.

GAME $\mathsf{ANO\text{-}PCA}_{\mathsf{KEM},b}^{(N,\mu),\mathcal{A}}$

$\mathsf{par} \leftarrow \mathsf{Setup}$
for $i \in [N]$
  $(\mathbf{pk}[i], \mathbf{sk}[i]) := (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}(\mathsf{par})$
  for $j \in [\mu]$:
    $(\mathsf{c}, \mathsf{k}) \leftarrow \mathsf{Encaps}(\mathbf{pk}[i])$
    $(\mathbf{c}_0[i,j], \mathbf{k}[i,j]) := (\mathsf{c}, \mathsf{k})$
    $\mathbf{c}_1[i,j] \xleftarrow{\$} \mathcal{C}$
$b' \leftarrow \mathcal{A}^{\mathrm{PCO}}(\mathsf{par}, \mathbf{pk}, \mathbf{c}_b)$
return b'

GAME $\mathsf{FUZZY}_{\mathsf{KEM},b}^{N,\mathcal{A}}$

$\mathsf{par} \leftarrow \mathsf{Setup}$
for $i \in [N]$
  $(\mathbf{pk}_0[i], \mathbf{sk}[i]) := (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}(\mathsf{par})$
  $\mathbf{pk}_1[i] := \mathsf{pk} \xleftarrow{\$} \mathcal{PK}$
$b' \leftarrow \mathcal{A}(\mathsf{par}, \mathbf{pk}_b)$
return b'

Fig. 3. Security games FUZZY and ANO-PCA for KEM scheme KEM. The PCO oracle of ANO-PCA is the same as the one of OW-PCA (and OW-rPCA) in Fig. 2.

Definition 5 (OW-PCA security under random ciphertexts). *Let N and μ be the number of users and the number of challenge ciphertexts per user, respectively. Let $\mathcal{A}$ be an adversary against* KEM. *We define the (N,μ)-OW-rPCA advantage function of $\mathcal{A}$*

$$\mathsf{Adv}_{\mathsf{KEM}}^{(N,\mu)\text{-}\mathsf{OW\text{-}rPCA}}(\mathcal{A}) := \Pr\left[\mathsf{OW\text{-}rPCA}_{\mathsf{KEM}}^{(N,\mu),\mathcal{A}} \Rightarrow 1\right],$$

where $\mathsf{OW\text{-}rPCA}_{\mathsf{KEM}}^{(N,\mu),\mathcal{A}}$ is defined in Fig. 2. KEM *is OW-rPCA secure if $\mathsf{Adv}_{\mathsf{KEM}}^{(N,\mu)\text{-}\mathsf{OW\text{-}rPCA}}(\mathcal{A})$ is negligible for any $\mathcal{A}$.*

Definition 6 (Fuzzy public keys). *Let N be the number of users. Let $\mathcal{A}$ be an adversary against* KEM. *We define the advantage function of $\mathcal{A}$ against the fuzzyness of* KEM

$$\mathsf{Adv}_{\mathsf{KEM}}^{N\text{-}\mathsf{FUZZY}}(\mathcal{A}) := \left|\Pr\left[\mathsf{FUZZY}_{\mathsf{KEM},0}^{N,\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\mathsf{FUZZY}_{\mathsf{KEM},1}^{N,\mathcal{A}} \Rightarrow 1\right]\right|,$$

where the game $\mathsf{FUZZY}^{N,\mathcal{A}}_{\mathsf{KEM},b}(b \in \{0,1\})$ *is defined in Fig. 3. We say* KEM *has fuzzy public keys if* $\mathsf{Adv}^{N\text{-}\mathsf{FUZZY}}_{\mathsf{KEM}}(\mathcal{A})$ *is negligible for any* $\mathcal{A}$.

Definition 7 (Anonymous ciphertexts under PCA attacks). *Let* N *and* μ *be the numbers of users and challenge ciphertexts per user, respectively. Let* $\mathcal{A}$ *be an adversary against* KEM. *We define the advantage function of* $\mathcal{A}$ *against the ciphertext anonymity (under PCA attacks) of* KEM

$$\mathsf{Adv}^{(N,\mu)\text{-}\mathsf{ANO}}_{\mathsf{KEM}}(\mathcal{A}) := \left|\Pr\left[\mathsf{ANO\text{-}PCA}^{(N,\mu),\mathcal{A}}_{\mathsf{KEM},0} \Rightarrow 1\right] - \Pr\left[\mathsf{ANO\text{-}PCA}^{(N,\mu),\mathcal{A}}_{\mathsf{KEM},1} \Rightarrow 1\right]\right|,$$

where the game $\mathsf{ANO\text{-}PCA}^{(N,\mu),\mathcal{A}}_{\mathsf{KEM},b}(b \in \{0,1\})$ *is defined in Fig. 3. We say* KEM *has anonymous ciphertexts under PCA attacks (or simply, anonymous ciphertexts) if* $\mathsf{Adv}^{(N,\mu)\text{-}\mathsf{ANO}}_{\mathsf{KEM}}(\mathcal{A})$ *is negligible for any* $\mathcal{A}$.

It is easy to see that if KEM is OW-PCA secure and has anonymous ciphertexts under PCA attacks, then it is also OW-rPCA secure, as stated in Lemma 1

Lemma 1 (OW-PCA + ANO-PCA ⇒ OW-rPCA). *Let* N *and* μ *be the numbers of users and challenge ciphertexts per user, respectively. Let* $\mathcal{A}$ *be an adversary against* KEM. *We have*

$$\mathsf{Adv}^{(N,\mu)\text{-}\mathsf{OW\text{-}rPCA}}_{\mathsf{KEM}}(\mathcal{A}) \le \mathsf{Adv}^{(N,\mu)\text{-}\mathsf{OW\text{-}PCA}}_{\mathsf{KEM}}(\mathcal{A}) + \mathsf{Adv}^{(N,\mu)\text{-}\mathsf{ANO}}_{\mathsf{KEM}}(\mathcal{A})$$

2.2 Public-Key Encryption

PUBLIC-KEY ENCRYPTION. A PKE scheme PKE consists of four algorithms $(\mathsf{Setup}, \mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ and a message space $\mathcal{M}$, a randomness space $\mathcal{R}$, and a ciphertext space $\mathcal{C}$. Setup outputs a system parameter par. $\mathsf{KG}(\mathsf{par})$ outputs a public and secret key pair $(\mathsf{pk}, \mathsf{sk})$. The encryption algorithm Enc, on input pk and a message $m \in \mathcal{M}$, outputs a ciphertext $\mathsf{c} \in \mathcal{C}$. We also write $\mathsf{c} := \mathsf{Enc}(\mathsf{pk}, m; r)$ to indicate the randomness $r \in \mathcal{R}$ explicitly. The decryption algorithm Dec, on input sk and a ciphertext c, outputs a message $m' \in \mathcal{M}$ or a rejection symbol $\perp \notin \mathcal{M}$.

Definition 8 (PKE Correctness). *Let* $\mathsf{PKE} := (\mathsf{Setup}, \mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ *be a PKE scheme with message space* $\mathcal{M}$ *and* $\mathcal{A}$ *be an adversary against* PKE. *The* COR *advantage of* $\mathcal{A}$ *is defined as*

$$\mathsf{Adv}^{\mathsf{COR}}_{\mathsf{PKE}}(\mathcal{A}) := \Pr\left[\mathsf{COR}^{\mathcal{A}}_{\mathsf{PKE}} \Rightarrow 1\right],$$

where the COR game is defined in Fig. 4. If there exists a constant δ *such that for all adversary* $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{COR}}_{\mathsf{PKE}}(\mathcal{A}) \le \delta$, *then we say* PKE *is* $(1-\delta)$*-correct.*

We define fuzzyness for PKE, which is essentially the same as the one for KEM (cf. Definition 6).

GAME $\mathsf{COR}^{\mathcal{A}}_{\mathsf{PKE}}$

$\mathsf{par} \leftarrow \mathsf{Setup}$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}(\mathsf{par})$
$m \leftarrow \mathcal{A}^{O}(\mathsf{par}, \mathsf{pk}, \mathsf{sk})$
$\mathsf{c} \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$
if $\mathsf{Dec}(\mathsf{sk}, \mathsf{c}) \neq m$: **return** 1
return 0

Fig. 4. The COR game for a PKE scheme PKE and $\mathcal{A}$. $\mathcal{A}$ might have access to some oracle O (e.g., random oracles). It depends on the specific reduction.

Definition 9 (Fuzzy public key). *Let N be the number of users. We say* PKE *has fuzzy public keys if for any $\mathcal{A}$, the advantage function of $\mathcal{A}$ against the fuzzyness of* PKE

$$\mathsf{Adv}^{N\text{-}\mathsf{FUZZY}}_{\mathsf{PKE}}(\mathcal{A}) := \left|\Pr\left[\mathsf{FUZZY}^{N,\mathcal{A}}_{\mathsf{PKE},0} \Rightarrow 1\right] - \Pr\left[\mathsf{FUZZY}^{N,\mathcal{A}}_{\mathsf{PKE},1} \Rightarrow 1\right]\right|$$

is negligible. The game $\mathsf{FUZZY}^{N,\mathcal{A}}_{\mathsf{PKE},b}(b \in \{0,1\})$ is defined in Fig. 3.

PSEUDORANDOM CIPHERTEXT. Let $\mathsf{PKE} := (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ be a public-key encryption scheme with message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$. We define PR-CPA (multi-challenge pseudorandomness under chosen-plaintext attacks) security in Fig. 5.

Definition 10 (Multi-user-challange PR-CPA security). *Let N and μ be the numbers of users and challenge ciphertexts per user. Let $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ be an adversary against* PKE. *Consider the games $\mathsf{PR\text{-}CPA}^{(N,\mu),\mathcal{A}}_{\mathsf{PKE},b}$ $(b \in \{0,1\})$ defined in Fig. 5. We define the (N,μ)-PR-CPA advantage function*

$$\mathsf{Adv}^{(N,\mu)\text{-}\mathsf{PR\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{A}) := \left|\Pr\left[\mathsf{PR\text{-}CPA}^{(N,\mu),\mathcal{A}}_{\mathsf{PKE},0} \Rightarrow 1\right] - \Pr\left[\mathsf{PR\text{-}CPA}^{(N,\mu),\mathcal{A}}_{\mathsf{PKE},1} \Rightarrow 1\right]\right|.$$

PKE *is PR-CPA secure if $\mathsf{Adv}^{(N,\mu)\text{-}\mathsf{PR\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{A})$ is negligible for any $\mathcal{A}$.*

3 Password-Based Authenticated Key Exchange

3.1 Definition of PAKE

A two-message PAKE protocol $\mathsf{PAKE} := (\mathsf{Setup}, \mathsf{Init}, \mathsf{Resp}, \mathsf{TerInit})$ consists of four algorithms. The setup algorithm Setup, on input security parameter 1^λ, outputs global PAKE protocol parameters par. For simplicity, we ignore the input of Setup and write $\mathsf{par} \leftarrow \mathsf{Setup}$.

Let U be a user, S be a server, and pw be the password shared between U and S. Since we consider the client-server setting, to initiate a session, U will send the first protocol message. U runs the client's initialization algorithm Init,

GAME $\mathsf{PR\text{-}CPA}_{\mathsf{PKE},b}^{(N,\mu),\mathcal{A}}$

$\mathsf{par} \leftarrow \mathsf{Setup}$
for $i \in N$
  $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KG}(\mathsf{par}), \mathbf{pk}[i] := \mathsf{pk}_i$
$(\mathbf{m}, \mathsf{st}) \leftarrow \mathcal{A}_0(\mathsf{par}, \mathbf{pk})$ // $\mathbf{m}$ has $N \times \mu$ messages
for $i \in [N]$:
  for $j \in [\mu]$
    $\mathbf{c}_0[i,j] \leftarrow \mathsf{Enc}(\mathbf{pk}[i], \mathbf{m}[i,j]), \mathbf{c}_1[i,j] \xleftarrow{\$} \mathcal{C}$
$b' \leftarrow \mathcal{A}_1(\mathsf{st}, \mathbf{c}_b)$
return b'

Fig. 5. Security game PR-CPA for PKE scheme PKE.

which takes the identities U, S and password pw as inputs and outputs a client message $\mathsf{M_U}$ and session state st, and then U sends $\mathsf{M_U}$ to S. On receiving $\mathsf{M_U}$, S runs the server's derivation algorithm Resp, which takes identities U and S and the received message $\mathsf{M_U}$ as input, together with the password pw, to generate a server message $\mathsf{M_S}$ and a session key $\mathsf{SK_S}$. S sends $\mathsf{M_S}$ to U. Finally, on receiving $\mathsf{M_S}$, U runs the client's derivation algorithm $\mathsf{TerInit}$ which inputs U, S, the session state st generated before, the received message $\mathsf{M_S}$, and password pw, to generate a session key $\mathsf{sk'_U}$. In two-message PAKE protocols, the server does not need to save session state since it can compute the session key right after receiving the user's message.

$$\begin{array}{lcl}
\text{User } \mathsf{U}(\mathsf{pw}) & & \text{Server } \mathsf{S}(\mathsf{pw}) \\
\hline
(\mathsf{M_U}, \mathsf{st}) \leftarrow \mathsf{Init}(\mathsf{U}, \mathsf{S}, \mathsf{pw}) & \xrightarrow{\mathsf{M_U}} & \\
\qquad \downarrow \mathsf{st} & & (\mathsf{M_S}, \mathsf{SK_S}) \leftarrow \mathsf{Resp}(\mathsf{S}, \mathsf{U}, \mathsf{M_U}, \mathsf{pw}) \\
\mathsf{SK_U} \leftarrow \mathsf{TerInit}(\mathsf{U}, \mathsf{S}, \mathsf{st}, \mathsf{M_S}, \mathsf{pw}) & \xleftarrow{\mathsf{M_S}} &
\end{array}$$

Fig. 6. Illustration for a two-message PAKE protocol execution between a user U and a server S.

We define the correctness of PAKE protocols, stating that an honestly execution between user U and server S (with the same password $\mathsf{pw_{U,S}}$) as in Fig. 6 will produce the same session key $\mathsf{SK_U} = \mathsf{SK_S}$.

Definition 11 (PAKE Correctness). *Let* $\mathsf{PAKE} := (\mathsf{Setup}, \mathsf{Init}, \mathsf{Resp}, \mathsf{TerInit})$ *be a PAKE protocol and let* U *and* S *be a user-server pair with password* pw*. We say* PAKE *is ρ-correct, if for any PAKE system parameter* $\mathsf{par} \leftarrow \mathsf{Setup}$*, the following probability is at least ρ.*

$$\Pr\left[\mathsf{SK_U} = \mathsf{SK_S} \;\middle|\; \begin{array}{c} (\mathsf{M_U}, \mathsf{st}) \leftarrow \mathsf{Init}(\mathsf{U}, \mathsf{S}, \mathsf{pw}) \\ (\mathsf{M_S}, \mathsf{SK_S}) \leftarrow \mathsf{Resp}(\mathsf{S}, \mathsf{U}, \mathsf{M_U}, \mathsf{pw}) \\ \mathsf{SK_U} \leftarrow \mathsf{TerInit}(\mathsf{U}, \mathsf{S}, \mathsf{st}, \mathsf{M_S}, \mathsf{pw}) \end{array}\right]$$

3.2 Security Model of PAKE

We consider indistinguishability(IND)-based security of PAKE protocols. In this section, we define the multi-$\texttt{test}$ variant of the Bellare-Pointcheval-Rogaway model [1,5,10]. We simply denoted it as the BPR model.

In the BPR model, we consider a name space of users $\mathcal{U}$ and a name space of servers $\mathcal{S}$, which are assumed to be disjoint. Oracles provided in this model rejects queries inconsistent withe these name spaces.

We denote the session key space by $\mathcal{SK}$. Password are bit strings of ℓ and the password space is defined as $\mathcal{PW} \subsetneq \{0,1\}^{\ell}$. Each pair of user and server $\mathsf{U} \times \mathsf{S} \in \mathcal{U} \times \mathcal{S}$ holds a shared password $\mathsf{pw}_{\mathsf{U},\mathsf{S}} \in \mathcal{PW}$.

Let P denotes a party (either a user or server). Each party in $\mathcal{U} \cup \mathcal{S}$ has multiple instances π_{P}^{i} (i is some index) and each instance has its internal state. The state of an instance π_{P}^{i} is a tuple $(\texttt{e}, \texttt{tr}, \texttt{key}, \texttt{acc})$ where

- $\texttt{e}$ is the ephemeral secret chosen by P.
- $\texttt{tr}$ is the trace of the instance, i.e., the names of user and server involved in the instance and the messages sent and received by P in the instance.
- $\texttt{key}$ is the accepted session key of π_{P}^{i}.
- $\texttt{acc}$ is a Boolean flag that indicates whether the instance has accepted the session key. As long as the instance did not receive the last message, $\texttt{acc} = \perp$ (which means undefined).
- $\texttt{test}$ is a Boolean flag that indicates whether the instance has been queried to the Test oracle (which will be defined later).

To access individual components of the state, we write $\pi_{\mathsf{P}}^{i}.(\texttt{e}, \texttt{tr}, \texttt{key}, \texttt{acc})$. We define partnership via matching instance trace.

Definition 12 (Partnering). *A user instance $\pi_{\mathsf{U}}^{t_0}$ and a server instance $\pi_{\mathsf{S}}^{t_1}$ are partnered if and only if*

$$\pi_{\mathsf{U}}^{t_0}.\texttt{acc} = \textbf{true} = \pi_{\mathsf{S}}^{t_1}.\texttt{acc} \quad \textbf{and} \quad \pi_{\mathsf{U}}^{t_0}.\texttt{tr} = \pi_{\mathsf{S}}^{t_1}.\texttt{tr}$$

Two user instances are never partnered, neither are two server instances. We define a partnership predicate $\mathsf{Partner}(\pi_{\mathsf{U}}^{t_0}, \pi_{\mathsf{S}}^{t_1})$ *which outputs* **true** *if and only if $\pi_{\mathsf{U}}^{t_0}$ and $\pi_{\mathsf{S}}^{t_1}$ are partnered.*

<u>Security Game.</u> The security game is played with an adversary $\mathcal{A}$. The experiment draws a random challenge bit $\beta \leftarrow \{0,1\}$, generates the public parameters, and outputs the public parameters to $\mathcal{A}$. $\mathcal{A}$ is allowed to query the following oracles:

- Execute$(\mathsf{U}, t_1, \mathsf{S}, t_2)$: This oracle outputs the protocol messages of an honest protocol execution between instances $\pi_{\mathsf{U}}^{t_1}$ and $\pi_{\mathsf{S}}^{t_2}$. By querying this oracle, the adversary launches passive attacks.

– SendInit, SendResp, SendTerInit: These oracles model active attacks. By querying these oracles, the adversary sends protocol messages to protocol instances. For sake of simplicity, we assume that the adversary does not use these oracles to launch passive attacks (which are already captured by the Execute oracle).
– Reveal(P, t): By this oracle, the adversary reveals the session key of π_{P}^t.
– Test(P, t): If π_{P}^t is fresh (which will be defined later), then, depending on the challenge bit β, the oracle outputs either the session key of π_{P}^t or a uniformly random key. Otherwise, the oracle outputs $\perp$. After this query, the flag $\pi_{\mathsf{P}}^t.\mathtt{test}$ will be set as **true**.

We denote the game by $\mathsf{BPR}_{\mathsf{PAKE}}$. The pseudocode is given in $\mathbf{G}_0$ in Fig. 8, instantiated with our PAKE protocol. Before defining PAKE security, we define freshness to avoid trivial attacks in this model.

Definition 13 (Freshness). *An instance π_{P}^t is fresh if and only if*

1. *π_{P}^t is accepted.*
2. *π_{P}^t was not queried to* Test *or* Reveal *before.*
3. *At least one of the following conditions holds:*
   (a) *π_{P}^t accepted during a query to* Execute*.*
   (b) *There exists more than one (not necessarily fresh) partner instance*[4]*.*
   (c) *A unique fresh partner instance exists.*
   (d) *No partner instance exists and the password of* P *was not corrupted prior to π_{P}^t is accepted.*

By these definitions, we are ready to define the security of PAKE protocols.

Definition 14 (Security of PAKE). *Let* PAKE *be a PAKE protocol and $\mathcal{A}$ be an adversary. The advantage of $\mathcal{A}$ against* PAKE *is defined as*

$$\mathsf{Adv}_{\mathsf{PAKE}}^{\mathsf{BPR}}(\mathcal{A}) := \left|\Pr\left[\mathsf{BPR}_{\mathsf{PAKE}}^{\mathcal{A}} \Rightarrow 1\right] - \frac{1}{2}\right|$$

A PAKE protocol is considered secure if the best the adversary can do is to perform an online dictionary attack. Concretely, PAKE *is secure if for any adversary $\mathcal{A}$, $\mathsf{Adv}_{\mathsf{PAKE}}^{\mathsf{BPR}}(\mathcal{A})$ is negligibly close to $\frac{S}{|\mathcal{PW}|}$ when passwords in the security game are drawn independently and uniformly from $\mathcal{PW}$. Here S is the number of send queries made by $\mathcal{A}$ (i.e., the number of sessions during the game $\mathsf{BPR}_{\mathsf{PAKE}}$).*

4 Our Generic Construction of PAKE

<u>Construction.</u> Let $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{KG}, \mathsf{Encaps}, \mathsf{Decaps})$ be a KEM scheme with public key space $\mathcal{PK}$, ciphertext space $\mathcal{C}$, and KEM key space $\mathcal{K}$. We also require KEM to have implicit rejection. Let $\mathsf{IC}_1 = (\mathsf{E}_1, \mathsf{D}_1)$ be a symmetric encryption with key space $\mathcal{PW}$, plaintext space $\mathcal{PK}$, and ciphertext space $\mathcal{E}_1$. Let $\mathsf{IC}_2 =$

Alg $\mathsf{Init}(\mathsf{U}, \mathsf{S}, \mathsf{pw})$	**Alg** $\mathsf{TerInit}(\mathsf{U}, \mathsf{S}, \mathsf{st}, e_2, \mathsf{pw})$	**Alg** $\mathsf{Resp}(\mathsf{S}, \mathsf{U}, e_1, \mathsf{pw})$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}(\mathsf{par})$	**let** $(\mathsf{pk}, \mathsf{sk}, e_1) := \mathsf{st}$	$\mathsf{pk} := \mathsf{D}_1(\mathsf{pw}, e_1)$
$e_1 := \mathsf{E}_1(\mathsf{pw}, \mathsf{pk})$	$\mathsf{c} := \mathsf{D}_2(\mathsf{pw}, e_2)$	$(\mathsf{c}, \mathsf{k}) \leftarrow \mathsf{Encaps}(\mathsf{pk})$
$\mathsf{st} := (\mathsf{pk}, \mathsf{sk}, e_1)$	$\mathsf{k} := \mathsf{Decaps}(\mathsf{sk}, \mathsf{c})$	$e_2 := \mathsf{E}_2(\mathsf{pw}, \mathsf{c})$
return (e_1, st)	$\mathrm{ctxt} := (\mathsf{U}, \mathsf{S}, e_1, e_2)$	$\mathrm{ctxt} := (\mathsf{U}, \mathsf{S}, e_1, e_2)$
	$\mathsf{SK} := \mathsf{H}(\mathrm{ctxt}, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw})$	$\mathsf{SK} := \mathsf{H}(\mathrm{ctxt}, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw})$
	return SK	**return** (e_2, SK)

Fig. 7. Our PAKE protocol Π.

$(\mathsf{E}_2, \mathsf{D}_2)$ be a symmetric encryption with key space $\mathcal{PW}$, plaintext space $\mathcal{C}$, and ciphertext space $\mathcal{E}_2$.

We construct our two-message PAKE protocol $\Pi = (\mathsf{Init}, \mathsf{Resp}, \mathsf{TerInit})$ as shown in Fig. 6, where $\mathcal{SK}$ is the session key space of PAKE and $\mathsf{H}\colon \{0,1\}^* \rightarrow \mathcal{SK}$ is a hash function which is used to derive the session key. The system parameter par is generated by $\mathsf{par} \leftarrow \mathsf{Setup}$.

The correctness of Π is dependent on KEM. In Fig. 7, one honest execution of Π includes one KEM encapsulation and decapsulation. So, if KEM is $(1-\delta)$-correct, then Π is also $(1-\delta)$-correct.

Theorem 1. *Let* H *be random oracle and* IC_1 *and* IC_2 *be ideal ciphers. If* KEM *is* $(1-\delta)$*-correct and has implicit rejection, fuzzy public keys, anonymous ciphertexts, OW-PCA security, and* $\mathsf{OW\text{-}rPCA}$ *security (cf. Definitions 4 to 7), then the PAKE protocol* Π *in Fig. 7 is secure (wrt Definition 14).*

Concretely, for any $\mathcal{A}$ *against* Π*, there are adversaries* $\mathcal{B}_1$*-*$\mathcal{B}_6$ *with* $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B}_i)(1 \leq i \leq 6)$ *and*

$$
\begin{aligned}
\mathsf{Adv}_{\Pi}^{\mathsf{BPR}}(\mathcal{A}) \leq \;& S/|\mathcal{PW}| + \mathsf{Adv}_{\mathsf{KEM}}^{q_1\text{-}\mathsf{FUZZY}}(\mathcal{B}_1) + \mathsf{Adv}_{\mathsf{KEM}}^{(S,q_2+S)\text{-}\mathsf{OW\text{-}rPCA}}(\mathcal{B}_4) \\
& + \mathsf{Adv}_{\mathsf{KEM}}^{(S,1)\text{-}\mathsf{OW\text{-}PCA}}(\mathcal{B}_2) + \mathsf{Adv}_{\mathsf{KEM}}^{(S+q_2,S)\text{-}\mathsf{OW\text{-}PCA}}(\mathcal{B}_5) \\
& + \mathsf{Adv}_{\mathsf{KEM}}^{(S,1)\text{-}\mathsf{ANO}}(\mathcal{B}_3) + \mathsf{Adv}_{\mathsf{KEM}}^{(S+q_1,S)\text{-}\mathsf{ANO}}(\mathcal{B}_6) + S \cdot \delta \\
& + S^2(\eta_{pk} + \eta_{ct}) + \frac{(q_1^2 + S^2)}{|\mathcal{E}_1|} + \frac{(q_2^2 + S^2)}{|\mathcal{E}_2|} + \frac{q_1^2}{|\mathcal{PK}|} + \frac{q_2^2}{|\mathcal{C}|} + \frac{(q_{\mathsf{H}}^2 + S^2)}{|\mathcal{SK}|},
\end{aligned}
$$

where q_1, q_2, q_{H} *are the numbers of* $\mathcal{A}$ *queries to* $\mathsf{IC}_1, \mathsf{IC}_2$*, and* H *respectively.* S *is the number of sessions* $\mathcal{A}$ *established in the security game.* η_{pk} *and* η_{ct} *are the collision probabilities of* KG *and* Encaps*, respectively.*

Remark 1 (Implementation of Ideal Ciphers). The implementation of IC_1 and IC_2 depends on the concrete instantiation of the underlying KEM scheme KEM. Beguinet et al. provides an implementation if KEM is instantiated with the Kyber KEM [32] in [8, Section 5.2]. More implementation for group-based schemes and lattice-based schemes can be found in [31].

[4] This essentially forces a secure PAKE protocol not to have more than one partner instances.

Remark 2. We require KEM to have implicit rejection (cf. Definition 3) because this simplifies our security proof. More concretely, if the underlying KEM KEM has implicit rejection, then we only require OW-PCA security to finish our tight proof. Otherwise, we need the OW-PCVA (cf. [21, Definition 2.1]) security to detect whether the c is valid in the proof.

4.1 Proof of Theorem 1

Let $\mathcal{A}$ be an adversary against PAKE in the BPR game, where N is the number of parties. Every user-server pair $(\mathsf{U}, \mathsf{S}) \in \mathcal{U} \times \mathcal{S}$ is associated with a password $\mathsf{pw}_{\mathsf{U},\mathsf{S}}$. The game sequences $\mathbf{G}_0$-$\mathbf{G}_{12}$ of the proof are given in Figs. 8, 9, 11, 14.

During the game sequences in this proof, we exclude the collisions of outputs of KG and Encaps in EXECUTE, SENDINIT, SENDRESP, and SENDTERINIT. We also exclude the collisions of outputs of ideal ciphers and random oracle, i.e., $\mathsf{IC}_1 = (\mathsf{E}_1, \mathsf{D}_1)$, $\mathsf{IC}_2 = (\mathsf{E}_2, \mathsf{D}_2)$, and H. If such a collision happens at any time, then we abort the game. For readability, we do not explicitly define such collision events in the codes of games sequences.

By the assumption of Theorem 1, the collision probabilities of the outputs of KG and Encaps are η_{pk} and η_{ct}, and S is the number of sessions generated (i.e., the total number of queries to EXECUTE, SENDINIT, SENDRESP, and SENDTERINIT) during the game and q_1, q_2, and q_{H} are the numbers of queries to IC_1, IC_2, and H, respectively. By birthday bounds and union bounds, such collision events happen within probability $S^2(\eta_{pk} + \eta_{ct}) + \frac{(q_1^2+S^2)}{|\mathcal{E}_1|} + \frac{(q_2^2+S^2)}{|\mathcal{E}_2|} + \frac{q_1^2}{|\mathcal{PK}|} + \frac{q_2^2}{|\mathcal{C}|} + \frac{(q_{\mathsf{H}}^2+S^2)}{|\mathcal{SK}|}$. Game $\mathbf{G}_0$ is the same as $\mathsf{BPR}_{\mathsf{PAKE}}$ except that we define such collision events in $\mathbf{G}_0$, we have

$$\begin{aligned}
&\left|\Pr\left[\mathsf{BPR}_{\mathsf{PAKE}}^{\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\mathbf{G}_0^{\mathcal{A}} \Rightarrow 1\right]\right| \\
&\leq S^2(\eta_{pk} + \eta_{ct}) + \frac{(q_1^2 + S^2)}{|\mathcal{E}_1|} + \frac{(q_2^2 + S^2)}{|\mathcal{E}_2|} + \frac{q_1^2}{|\mathcal{PK}|} + \frac{q_2^2}{|\mathcal{C}|} + \frac{(q_{\mathsf{H}}^2 + S^2)}{|\mathcal{SK}|}
\end{aligned}$$

Moreover, excluding these collisions imply that different instances have different traces and each instance (user's or server's) has at most one partnering instance. By the construction of PAKE, different instances will have different session keys, since the hash function H take the trace of instance as input.

Game $\mathbf{G}_1$. Instead of using the $\mathsf{Freshness}$ procedure in the TEST oracle, we assign an additional variable `fr` to each instance π to explicitly indicate the freshness of π. Whenever $\mathcal{A}$ issues an oracle query related to π, we will update π.`fr` in real time according to the freshness definition (cf. Definition 13). This change is conceptual, so we have

$$\Pr\left[\mathbf{G}_0^{\mathcal{A}} \Rightarrow 1\right] = \Pr\left[\mathbf{G}_1^{\mathcal{A}} \Rightarrow 1\right]$$

To save space, for games $\mathbf{G}_2$ to $\mathbf{G}_x$, instead of presenting the whole codes of the game, we only present the codes of changed oracles.

Game $\mathbf{G}_0$-$\mathbf{G}_1$

$\mathsf{par} \leftarrow \mathsf{Setup}$
for $(\mathsf{U},\mathsf{S}) \in \mathcal{U} \times \mathcal{S}$
  $\mathsf{pw}_{\mathsf{U},\mathsf{S}} \leftarrow \mathcal{PW}$
$\mathcal{C} := \emptyset$
$\beta \leftarrow \{0,1\}$
$b' \leftarrow \mathcal{A}^{O,\mathsf{H},\mathsf{IC}_1,\mathsf{IC}_2}(\mathsf{par})$
return $\beta == b'$

Oracle $\textsc{Reveal}(\mathsf{P}, t)$

if $\pi_{\mathsf{P}}^{t}.\mathtt{acc} \neq$ **true or** $\pi_{\mathsf{P}}^{t}.\mathtt{test} =$ **true**
  return $\perp$
if $\exists \mathsf{P}' \in \mathcal{U} \cup \mathcal{S}, t'$ s.t.
  $\mathsf{Partner}(\pi_{\mathsf{P}}^{t}, \pi_{\mathsf{P}'}^{t'}) =$ **true**
  and $\pi_{\mathsf{P}'}^{t'}.\mathtt{test} =$ **true**
  return $\perp$
for $\forall(\mathsf{P}', t')$ s.t. $\pi_{\mathsf{P}'}^{t'}.\mathtt{tr} = \pi_{\mathsf{P}}^{t}.\mathtt{tr}$ // $\mathbf{G}_1$
  $\pi_{\mathsf{P}'}^{t'}.\mathtt{fr} :=$ **false** // $\mathbf{G}_1$
return $\pi_{\mathsf{P}}^{t}.\mathtt{key}$

Oracle $\textsc{Test}(\mathsf{P}, t)$

if $\mathsf{Freshness}(\pi_{\mathsf{P}}^{t}) =$ **false** // $\mathbf{G}_0$
if $\pi_{\mathsf{P}}^{t}.\mathtt{fr} =$ **false** // $\mathbf{G}_1$
  return $\perp$
$\mathsf{SK}_0^* := \textsc{Reveal}(\mathsf{P}, t)$, $\mathsf{SK}_1^* \xleftarrow{\$} \mathcal{SK}$
if $\mathsf{SK}_0^* = \perp$: **return** $\perp$
$\pi_{\mathsf{P}}^{t}.\mathtt{test} :=$ **true**
return SK_β^*

Oracle $\textsc{Corrupt}(\mathsf{U}, \mathsf{S})$

if $(\mathsf{U},\mathsf{S}) \in \mathcal{C}$: **return** $\perp$
$\mathcal{C} := \mathcal{C} \cup \{(\mathsf{U},\mathsf{S})\}$
return $\mathsf{pw}_{\mathsf{U},\mathsf{S}}$

Oracle $\mathsf{E}_1(\mathsf{pw}, \mathsf{pk})$

if $\exists(\mathsf{pw}, \mathsf{pk}, e_1, *) \in \mathcal{L}_1$: **return** e_1
$e_1 \xleftarrow{\$} \mathcal{E}_1 \backslash \mathcal{T}_1$, $\mathcal{L}_1 := \mathcal{L}_1 \cup \{e_1\}$
$\mathcal{L}_1 := \mathcal{L}_1 \cup (\mathsf{pw}, \mathsf{pk}, e_1, \mathrm{enc})$
return e_1

Oracle $\mathsf{E}_2(\mathsf{pw}, \mathsf{c})$

if $\exists(\mathsf{pw}, \mathsf{c}, e_2, *) \in \mathcal{L}_2$: **return** e_2
$e_2 \xleftarrow{\$} \mathcal{E}_2 \backslash \mathcal{T}_2$, $\mathcal{T}_2 := \mathcal{T}_2 \cup \{e_2\}$
$\mathcal{L}_2 := \mathcal{L}_2 \cup (\mathsf{pw}, \mathsf{c}, e_2, \mathrm{enc})$
return e_2

Oracle $\mathsf{D}_1(\mathsf{pw}, e_1)$

if $\exists(\mathsf{pw}, \mathsf{pk}, e_1, *) \in \mathcal{L}_1$: **return** pk
$\mathsf{pk} \xleftarrow{\$} \mathcal{PK}$, $\mathcal{L}_1 := \mathcal{L}_1 \cup (\mathsf{pw}, \mathsf{pk}, e_1, \mathrm{dec})$
return pk

Oracle $\mathsf{D}_2(\mathsf{pw}, e_2)$

if $\exists(\mathsf{pw}, \mathsf{c}, e_2, *) \in \mathcal{L}_2$: **return** c
$\mathsf{c} \xleftarrow{\$} \mathcal{C}$, $\mathcal{L}_2 := \mathcal{L}_2 \cup (\mathsf{pw}, \mathsf{c}, e_2, \mathrm{dec})$
return c

Oracle $\textsc{Execute}(\mathsf{U}, t_1, \mathsf{S}, t_2)$

if $\pi_{\mathsf{U}}^{t_1} \neq \perp$ **or** $\pi_{\mathsf{S}}^{t_2} \neq \perp$
  return $\perp$
let $\mathsf{pw} := \mathsf{pw}_{\mathsf{U},\mathsf{S}}$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}(\mathsf{par})$, $e_1 := \mathsf{E}_1(\mathsf{pw}, \mathsf{pk})$
$(\mathsf{c}, \mathsf{k}) \leftarrow \mathsf{Encaps}(\mathsf{pk})$, $e_2 := \mathsf{E}_2(\mathsf{pw}, \mathsf{c})$
$\mathsf{ctxt} := (\mathsf{U}, \mathsf{S}, e_1, e_2)$
$\mathsf{SK} := \mathsf{H}(\mathsf{ctxt}, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw})$
$\pi_{\mathsf{U}}^{t_1} := ((\mathsf{pk}, \mathsf{sk}, e_1), \mathsf{ctxt}, \mathsf{SK}, \mathbf{true})$
$\pi_{\mathsf{S}}^{t_2} := ((\mathsf{c}, \mathsf{k}, e_2), \mathsf{ctxt}, \mathsf{SK}, \mathbf{true})$
$(\pi_{\mathsf{U}}^{t_1}.\mathtt{fr}, \pi_{\mathsf{S}}^{t_2}.\mathtt{fr}) := (\mathbf{true}, \mathbf{true})$ // $\mathbf{G}_1$
return $(\mathsf{U}, e_1, \mathsf{S}, e_2)$

Oracle $\textsc{SendInit}(\mathsf{U}, t_1, \mathsf{S})$

if $\pi_{\mathsf{U}}^{t_1} \neq \perp$: **return** $\perp$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}(\mathsf{par})$
$e_1 := \mathsf{E}_1(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, \mathsf{pk})$
$\pi_{\mathsf{U}}^{t_1} := ((\mathsf{pk}, \mathsf{sk}, e_1), (\mathsf{U}, \mathsf{S}, e_1, \perp), \perp, \perp)$
$\pi_{\mathsf{U}}^{t_1}.\mathtt{fr} :=$ **false** // $\mathbf{G}_1$
return (U, e_1)

Oracle $\textsc{SendResp}(\mathsf{S}, t_2, \mathsf{U}, e_1)$

$\pi_{\mathsf{S}}^{t_2} \neq \perp$: **return** $\perp$
if $(\mathsf{U},\mathsf{S}) \in \mathcal{C}$: $\pi_{\mathsf{S}}^{t_2}.\mathtt{fr} :=$ **false** // $\mathbf{G}_1$
else $\pi_{\mathsf{S}}^{t_2}.\mathtt{fr} :=$ **true** // $\mathbf{G}_1$
$\mathsf{pk} := \mathsf{D}_1(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, e_1)$
$(\mathsf{c}, \mathsf{k}) \leftarrow \mathsf{Encaps}(\mathsf{pk})$
$e_2 := \mathsf{E}_2(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, \mathsf{c})$
$\mathsf{ctxt} := (\mathsf{U}, \mathsf{S}, e_1, e_2)$
$\mathsf{SK} := \mathsf{H}(\mathsf{ctxt}, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw}_{\mathsf{U},\mathsf{S}})$
$\pi_{\mathsf{S}}^{t_2} := ((\mathsf{c}, \mathsf{k}, e_2), \mathsf{ctxt}, \mathsf{SK}, \mathbf{true})$
return (S, e_2)

Oracle $\textsc{SendTerInit}(\mathsf{U}, t_1, \mathsf{S}, e_2)$

if $\pi_{\mathsf{U}}^{t_1} = \perp$ **and** $\pi_{\mathsf{U}}^{t_1}.\mathtt{tr} \neq (\mathsf{U}, \mathsf{S}, *, *)$
  return $\perp$
let $(\mathsf{pk}, \mathsf{sk}, e_1) := \pi_{\mathsf{U}}^{t_1}.\mathtt{e}$
$\mathsf{c} := \mathsf{D}_2(\mathsf{pw}, e_2)$, $\mathsf{k} := \mathsf{Decaps}(\mathsf{sk}, \mathsf{c})$
if $\exists t_2$ s.t. $\pi_{\mathsf{S}}^{t_2}.\mathtt{fr} =$ **true** // $\mathbf{G}_1$
  and $\pi_{\mathsf{S}}^{t_2}.\mathtt{tr} = (\mathsf{U}, \mathsf{S}, e_1, e_2)$ // $\mathbf{G}_1$
  $\pi_{\mathsf{U}}^{t_1}.\mathtt{fr} :=$ **true** // $\mathbf{G}_1$
else if $(\mathsf{U},\mathsf{S}) \notin \mathcal{C}$: $\pi_{\mathsf{U}}^{t_1}.\mathtt{fr} :=$ **true** // $\mathbf{G}_1$
else $\pi_{\mathsf{U}}^{t_1}.\mathtt{fr} :=$ **false** // $\mathbf{G}_1$
$\mathsf{ctxt} := (\mathsf{U}, \mathsf{S}, e_1, e_2)$
$\mathsf{SK} := \mathsf{H}(\mathsf{ctxt}, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw}_{\mathsf{U},\mathsf{S}})$
$\pi_{\mathsf{U}}^{t_1}.(\mathtt{tr}, \mathtt{key}, \mathtt{acc}) := (\mathsf{ctxt}, \mathsf{SK}, \mathbf{true})$
return true

Oracle $\mathsf{H}(\mathsf{U}, \mathsf{S}, e_1, e_2, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw})$

if $\mathcal{L}_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, e_1, e_2, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw}] = \perp$
  $\mathsf{SK} \xleftarrow{\$} \mathcal{SK}$
  $\mathcal{L}_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, e_1, e_2, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw}] := \mathsf{SK}$
return $\mathcal{L}_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, e_1, e_2, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw}]$

Fig. 8. Games in proving Theorem 1. $\mathcal{A}$ has access to the set of PAKE oracles $\{\textsc{Execute}, \textsc{SendInit}, \textsc{SendResp}, \textsc{SendTerInit}, \textsc{Corrupt}, \textsc{Reveal}, \textsc{Test}\}$, random oracle H, and ideal ciphers $\mathsf{IC}_1 = (\mathsf{E}_1, \mathsf{D}_1)$ and $\mathsf{IC}_2 = (\mathsf{E}_2, \mathsf{D}_2)$.

Oracle $\textsc{Execute}(\mathsf{U}, t_1, \mathsf{S}, t_2)$

01 **if** $\pi_{\mathsf{U}}^{t_1} \neq \perp$ **or** $\pi_{\mathsf{S}}^{t_2} \neq \perp$
02   **return** $\perp$
03 $\mathsf{pw} := \mathsf{pw}_{\mathsf{U},\mathsf{S}}$
04 $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}(\mathsf{par}), e_1 := \mathsf{E}_1(\mathsf{pw}, \mathsf{pk})$ // $\mathbf{G}_1$-$\mathbf{G}_4$
05 $(\mathsf{c}, \mathsf{k}) \leftarrow \mathsf{Encaps}(\mathsf{pk}), e_2 := \mathsf{E}_2(\mathsf{pw}, \mathsf{c})$ // $\mathbf{G}_1$-$\mathbf{G}_3$
06 $\mathsf{c} \xleftarrow{\$} \mathcal{C}, e_2 := \mathsf{E}_2(\mathsf{pw}, \mathsf{c})$ // $\mathbf{G}_4$
07 $e_1 \xleftarrow{\$} \mathcal{E}_1 \backslash \mathcal{T}_1, \mathcal{T}_1 := \mathcal{T}_1 \cup \{e_1\}$ // $\mathbf{G}_5$
08 $e_2 \xleftarrow{\$} \mathcal{E}_2 \backslash \mathcal{T}_2, \mathcal{T}_2 := \mathcal{T}_2 \cup \{e_2\}$ // $\mathbf{G}_5$
09 $\mathrm{ctxt} := (\mathsf{U}, \mathsf{S}, e_1, e_2)$
10 $\mathsf{SK} := \mathsf{H}(\mathrm{ctxt}, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw})$ // $\mathbf{G}_1$-$\mathbf{G}_2$
11 $\mathsf{SK} \xleftarrow{\$} \mathcal{SK}$ // $\mathbf{G}_3$-$\mathbf{G}_5$
12 $\pi_{\mathsf{U}}^{t_1} := ((\mathsf{pk}, \mathsf{sk}, e_1), \mathrm{ctxt}, \mathsf{SK}, \mathbf{true})$ // $\mathbf{G}_1$-$\mathbf{G}_3$
13 $\pi_{\mathsf{S}}^{t_2} := ((\mathsf{c}, \mathsf{k}, e_2), \mathrm{ctxt}, \mathsf{SK}, \mathbf{true})$ // $\mathbf{G}_1$-$\mathbf{G}_3$
14 $\pi_{\mathsf{U}}^{t_1} := ((\perp, \perp, e_1), \mathrm{ctxt}, \mathsf{SK}, \mathbf{true})$ // $\mathbf{G}_4$-$\mathbf{G}_5$
15 $\pi_{\mathsf{S}}^{t_2} := ((\perp, \perp, e_2), \mathrm{ctxt}, \mathsf{SK}, \mathbf{true})$ // $\mathbf{G}_4$-$\mathbf{G}_5$
16 $(\pi_{\mathsf{U}}^{t_1}.\mathtt{fr}, \pi_{\mathsf{S}}^{t_2}.\mathtt{fr}) := (\mathbf{true}, \mathbf{true})$
17 **return** $(\mathsf{U}, e_1, \mathsf{S}, e_2)$

Oracle $\mathsf{D}_1(\mathsf{pw}, e_1)$

18 **if** $\exists(\mathsf{pw}, \mathsf{pk}, e_1, *) \in \mathcal{L}_1$
19   **return** pk
20 $\mathsf{pk} \xleftarrow{\$} \mathcal{PK}$ // $\mathbf{G}_1$
21 $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}$ // $\mathbf{G}_2$-$\mathbf{G}_5$
22 $\mathcal{L}_{\mathsf{key}} := \mathcal{L}_{\mathsf{key}} \cup \{(\mathsf{pk}, \mathsf{sk})\}$ // $\mathbf{G}_2$-$\mathbf{G}_5$
23 $\mathcal{L}_1 := \mathcal{L}_1 \cup \{(\mathsf{pw}, \mathsf{pk}, e_1, \mathrm{dec})\}$
24 **return** pk

Fig. 9. Oracles $\textsc{Execute}$ and D_1 in the games sequence $\mathbf{G}_1$-$\mathbf{G}_5$.

Game $\mathbf{G}_2$. We change the output of D_1. When $\mathcal{A}$ queries $\mathsf{D}_1(\mathsf{pw}, e_1)$ where e_1 is not generated from $\mathsf{E}_1(\mathsf{pw}, \cdot)$, we generate pk via $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}$ instead of $\mathsf{pk} \xleftarrow{\$} \mathcal{PK}$. Such $(\mathsf{pk}, \mathsf{sk})$ is recorded in $\mathcal{L}_{\mathsf{key}}$. cf. Lines 20 ro 22.

The difference between $\mathbf{G}_1$ and $\mathbf{G}_2$ can be bounded by using the fuzzyness of KEM. The bound is given in Lemma 2. For readability, we continue the proof of Lemma 1 and postpone the proof of Lemma 2 to our full version [28].

Lemma 2. *With notations and assumptions from $\mathbf{G}_1$ and $\mathbf{G}_2$ in the proof of Theorem 1, there is an adversary $\mathcal{B}_1$ with $\mathbf{T}(\mathcal{B}_1) \approx \mathbf{T}(\mathcal{A})$ and*

$$\left|\Pr\left[\mathbf{G}_1^{\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\mathbf{G}_2^{\mathcal{A}} \Rightarrow 1\right]\right| \leq \mathsf{Adv}_{\mathsf{KEM}}^{q_1\text{-}\mathsf{FUZZY}}(\mathcal{B}_1)$$

After this change, all pk generated by querying D_1 (i.e., there exists (pw, e_1) s.t. $(\mathsf{pw}, \mathsf{pk}, e_1, \mathrm{dec}) \in \mathcal{L}_1$) will always have a secret key sk such that $(\mathsf{pk}, \mathsf{sk}) \in \mathcal{L}_{\mathsf{key}}$. This fact is crucial for our later simulation.

Game $\mathbf{G}_3$. In this game, session keys of instances generated in $\textsc{Execute}$ are all uniformly at random and independent of H (cf. Lines 10 to 11).

Let $\mathsf{Query}_{\mathrm{exec}}$ be the event that $\mathcal{A}$ queries the hash input of the session key of an instance generated in $\textsc{Execute}$. Since H is a random oracle, if $\mathsf{Query}_{\mathrm{exec}}$ does not happen, then $\mathcal{A}$ cannot detect the modification made in $\mathbf{G}_3$. We have

$$\left|\Pr\left[\mathbf{G}_2^{\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\mathbf{G}_3^{\mathcal{A}} \Rightarrow 1\right]\right| \leq \Pr\left[\mathsf{Query}_{\mathrm{exec}}\right]$$

We construct an adversary $\mathcal{B}_2$ against the OW-PCA security of KEM in Fig. 10 such that $\mathbf{T}(\mathcal{B}_2) \approx \mathbf{T}(\mathcal{A})$ and $\Pr[\mathsf{Query}_{\text{exec}}] \leq \mathsf{Adv}_{\mathsf{KEM}}^{(S,1)\text{-}\mathsf{OW}\text{-}\mathsf{PCA}}(\mathcal{B}_2)$. Concretely, $\mathcal{B}_2$ inputs a OW-PCA challenge $(\mathsf{par}, \mathbf{pk}, \mathbf{c})$ and has access to a plaintext checking oracle PCO. Since $\mathcal{A}$'s number of queries to EXECUTE is S and there is only one KEM ciphertext generated per query to EXECUTE, we need at most S challenge public keys and one challenge ciphertexts per public key.

Reduction $\mathcal{B}_2^{\mathrm{PCO}(\cdot,\cdot,\cdot)}(\mathsf{par}, \mathbf{pk}, \mathbf{c})$

01 $\mathsf{cnt} := 0, \mathcal{L}_{\mathrm{E}} := \emptyset$
02 $i^* := \perp, j^* := \perp, \mathsf{k}^* := \perp$
03 $\mathsf{Query}_{\text{exec}} := \mathbf{false}$
04 **for** $(\mathsf{U}, \mathsf{S}) \in \mathcal{U} \times \mathcal{S}$
05 $\quad \mathsf{pw}_{\mathsf{U},\mathsf{S}} \leftarrow \mathcal{PW}$
06 $\mathcal{C} := \emptyset, \beta \leftarrow \{0,1\}$
07 $b' \leftarrow \mathcal{A}^{\mathrm{O},\mathsf{H},\mathsf{IC}_1,\mathsf{IC}_2}(\mathsf{par})$
08 **return** (i^*, j^*, k^*)

Oracle $\mathsf{H}(\mathsf{U}, \mathsf{S}, e_1, e_2, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw})$

09 $\mathsf{ctxt} := (\mathsf{U}, \mathsf{S}, e_1, e_2)$
10 **if** $\exists i'$ s.t. $(\mathsf{ctxt}, (\mathsf{pk}, i'), \mathsf{c}, \mathsf{pw}) \in \mathcal{L}_{\mathrm{E}}$
11 $\quad$ **and** $\mathrm{PCO}(\mathsf{cnt}^*, \mathsf{c}, \mathsf{k}) = 1$
12 $\quad \mathsf{Query}_{\text{exec}} := \mathbf{true}$
13 $\quad (i^*, j^*, \mathsf{k}^*) := (i', 1, \mathsf{k})$
14 **if** $\mathcal{L}_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, e_1, e_2, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw}] = \perp$
15 $\quad \mathsf{SK} \xleftarrow{\$} \mathcal{SK}$
16 $\quad \mathcal{L}_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, e_1, e_2, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw}] := \mathsf{SK}$
17 **return** $\mathcal{L}_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, e_1, e_2, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw}]$

Oracle $\mathrm{EXECUTE}(\mathsf{U}, t_1, \mathsf{S}, t_2)$

18 **if** $\pi_{\mathsf{U}}^{t_1} \neq \perp$ **or** $\pi_{\mathsf{S}}^{t_2} \neq \perp$
19 $\quad$ **return** $\perp$
20 $\mathsf{pw} := \mathsf{pw}_{\mathsf{U},\mathsf{S}}$, $\mathsf{cnt} := \mathsf{cnt} + 1$
21 $\mathsf{pk} := \mathbf{pk}[\mathsf{cnt}]$, $e_1 := \mathsf{E}_1(\mathsf{pw}, \mathsf{pk})$
22 $\mathsf{c} := \mathbf{c}[\mathsf{cnt}, 1]$, $e_2 := \mathsf{E}_2(\mathsf{pw}, \mathsf{c})$
23 $\mathsf{ctxt} := (\mathsf{U}, \mathsf{S}, e_1, e_2)$
24 $\mathcal{L}_{\mathrm{E}} := \mathcal{L}_{\mathrm{E}} \cup \{(\mathsf{ctxt}, (\mathsf{pk}, \mathsf{cnt}), \mathsf{c}, \mathsf{pw})\}$
25 $\mathsf{SK} \xleftarrow{\$} \mathcal{SK}$
26 $\pi_{\mathsf{U}}^{t_1} := ((\mathsf{pk}, \perp, e_1), \mathtt{tr}, \mathsf{SK}, \mathbf{true})$
27 $\pi_{\mathsf{S}}^{t_2} := ((\mathsf{c}, \perp, e_2), \mathtt{tr}, \mathsf{SK}, \mathbf{true})$
28 $(\pi_{\mathsf{U}}^{t_1}.\mathtt{fr}, \pi_{\mathsf{S}}^{t_2}.\mathtt{fr}) := (\mathbf{true}, \mathbf{true})$
29 **return** $(\mathsf{U}, e_1, \mathsf{S}, e_2)$

Fig. 10. Reduction $\mathcal{B}_2$ in bounding the probability difference between $\mathbf{G}_2$ and $\mathbf{G}_3$. Highlighted parts show how $\mathcal{B}_2$ uses PCO and challenge input to simulate $\mathbf{G}_3$. All other oracles (except EXECUTE and H) are the same as in $\mathbf{G}_2$.

$\mathcal{B}_2$ uses (i^*, j^*, k^*) to store its OW solution and uses $\mathcal{L}_{\mathrm{E}}$ to record the intended hash input of session keys generated in EXECUTE (cf. Line 24). Although $\mathcal{B}_2$ does not have secret keys of $\mathbf{pk}$ and KEM keys of $\mathbf{c}$, it can still simulate $\mathbf{G}_3$ since this information is not required in simulating EXECUTE. Moreover, $\mathcal{B}_2$ uses $\mathcal{L}_{\mathrm{E}}$ and PCO to determine whether $\mathsf{Query}_{\text{exec}}$ happens (cf. Lines 10 to 13).

If $\mathcal{A}$ queried $\mathsf{H}(\mathsf{U}, \mathsf{S}, e_1, e_2, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw})$, where $(\mathsf{U}, \mathsf{S}, e_1, e_2, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw})$ is the intended hash input of a session key SK generated in EXECUTE, then by the construction of PAKE and Lines 21 to 24, there exists $\mathsf{cnt}^* \in [S]$ such that $(\mathsf{U}, \mathsf{S}, e_1, e_2, (\mathsf{pk}, \mathsf{cnt}^*), \mathsf{c}, \mathsf{pw}) \in \mathcal{L}_{\mathrm{E}}$, $\mathsf{c} = \mathbf{c}[\mathsf{cnt}^*, 1]$, and $\mathsf{k} = \mathsf{Decaps}(\mathsf{sk}, \mathsf{c})$, where sk is the secret key of $\mathbf{pk}[\mathsf{cnt}^*]$. This means that k is the OW solution of $\mathbf{c}[\mathsf{cnt}^*, 1]$, and thus $\mathcal{B}_2$ records the OW solution (cf. Line 13) and returns it when the game ends. Therefore, we have

$$\left|\Pr\left[\mathbf{G}_2^{\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\mathbf{G}_3^{\mathcal{A}} \Rightarrow 1\right]\right| \leq \Pr\left[\mathsf{Query}_{\text{exec}}\right] \leq \mathsf{Adv}_{\mathsf{KEM}}^{(S,1)\text{-}\mathsf{OW}\text{-}\mathsf{PCA}}(\mathcal{B}_2).$$

Game $\mathbf{G}_4$. We change the generation of c in EXECUTE (cf. Line 06). In this game, c is sampled from $\mathcal{C}$ uniformly at random instead of using Encaps. Moreover, we no longer store the information about pk, sk, c, and k in the outputting instances from EXECUTE (cf. Lines 14 to 15). The later modification is conceptual since the game does not need this information to simulate EXECUTE.

The difference between $\mathbf{G}_3$ and $\mathbf{G}_4$ can be bounded by using the ciphertext anonymity of KEM. The bound is given in Lemma 3. We continue the proof of Theorem 1 and postpone the proof of Lemma 3 to our full version [28].

Lemma 3. *With notations and assumptions from $\mathbf{G}_3$ and $\mathbf{G}_4$ in the proof of Theorem 1, there is an adversary $\mathcal{B}_3$ with $\mathbf{T}(\mathcal{B}_3) \approx \mathbf{T}(\mathcal{A})$ and*

$$\left|\Pr\left[\mathbf{G}_3^{\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\mathbf{G}_4^{\mathcal{A}} \Rightarrow 1\right]\right| \leq \mathsf{Adv}_{\mathsf{KEM}}^{(S,1)\text{-}\mathsf{ANO}}(\mathcal{B}_3)$$

Game $\mathbf{G}_5$. We postpone the generation of pk and c in EXECUTE. Concretely, when $\mathcal{A}$ issues a query $(\mathsf{U}, t_1, \mathsf{S}, t_2)$ to EXECUTE, we sample e_1 and e_2 uniformly at random (cf. Lines 07 to 08) and postpone the generation of pk and c and usage of IC_1 and IC_2 to the time that $\mathcal{A}$ queries $\mathsf{D}_1(\mathsf{pw}_{\mathsf{U,S}}, e_1)$ or $\mathsf{D}_2(\mathsf{pw}_{\mathsf{U,S}}, e_2)$, respectively. The change made in $\mathbf{G}_2$ ensures that pk output by $\mathsf{D}_1(\mathsf{pw}_{\mathsf{U,S}}, e_1)$ is generated using KG, and the change made in $\mathbf{G}_4$ ensures that c output by $\mathsf{D}_2(\mathsf{pw}_{\mathsf{U,S}}, e_2)$ is generated via uniformly sampling over $\mathcal{C}$. Therefore, $\mathbf{G}_5$ is conceptually equivalent to $\mathbf{G}_4$, which means

$$\Pr\left[\mathbf{G}_4^{\mathcal{A}} \Rightarrow 1\right] = \Pr\left[\mathbf{G}_5^{\mathcal{A}} \Rightarrow 1\right]$$

Game $\mathbf{G}_6$. We rewrite the codes of SENDINIT, SENDRESP, and SENDTERINIT in Fig. 11. In this game, SENDRESP and SENDTERINIT compute session keys based on the freshness of instances. SENDRESP in $\mathbf{G}_6$ is equivalent to the one in $\mathbf{G}_5$. For SENDTERINIT in $\mathbf{G}_6$, if the user instance $\pi_{\mathsf{U}}^{t_1}$ has a matching server instance and such instance is fresh, then we make these two instances have the same session key (cf. Line 46). These changes are for further game transitions and they are conceptual if KEM has perfect correctness. Here we need to consider the correctness error of KEM since now we directly set up $\pi_{\mathsf{U}}^{t_1}$'s session key without decapsulation. There are at most S queries to SENDTERINIT, by a union bound, we have

$$\left|\Pr\left[\mathbf{G}_5^{\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\mathbf{G}_6^{\mathcal{A}} \Rightarrow 1\right]\right| \leq S \cdot \delta.$$

Game $\mathbf{G}_7$. We use two flags $\mathsf{Guess}_{\mathrm{user}}$ and $\mathsf{Guess}_{\mathrm{ser}}$ (which are initialized as **false**) to indicate whether the following events happen:

- When $\mathcal{A}$ queries SENDRESP$(\mathsf{S}, t_2, \mathsf{U}, e_1)$, if (U, S) is uncorrupted, e_1 is not generated from U's instance (cf. Line 37), and $\exists \mathsf{pk}$ such that e_1 is generated via querying $\mathsf{E}_1(\mathsf{pw}_{\mathsf{U,S}}, \mathsf{pk})$, then we set $\mathsf{Guess}_{\mathrm{ser}}$ as **true** (cf. Lines 23 to 24).
- When $\mathcal{A}$ queries SENDTERINIT$(\mathsf{U}, t_1, \mathsf{S}, e_2)$, if $\pi_{\mathsf{U}}^{t_1}$ does not have matching session, (U, S) is uncorrupted, e_2 is not generated from S's instance (cf. Line 30), and $\exists \mathsf{c}$ such that e_2 is generated via querying $\mathsf{E}_2(\mathsf{pw}_{\mathsf{U,S}}, \mathsf{c})$, then we set $\mathsf{Guess}_{\mathrm{user}}$ as **true** (cf. Lines 53 to 53).

Game $\mathbf{G}_6$-$\mathbf{G}_{10}$

$\mathsf{par} \leftarrow \mathsf{Setup}$
for $(\mathsf{U},\mathsf{S}) \in \mathcal{U}$: $\mathsf{pw}_{\mathsf{U},\mathsf{S}} \leftarrow \mathcal{PW}$
$\mathcal{C} := \emptyset, \beta \leftarrow \{0,1\}$
$\mathsf{Guess}_{\mathrm{user}} :=$ **false** // $\mathbf{G}_7$-$\mathbf{G}_{10}$
$\mathsf{Guess}_{\mathrm{ser}} :=$ **false** // $\mathbf{G}_7$-$\mathbf{G}_{10}$
$b' \leftarrow \mathcal{A}^{O,\mathsf{H},\mathsf{IC}_1,\mathsf{IC}_2}(\mathsf{par})$
return $\beta == b'$

Oracle $\textsc{SendResp}(\mathsf{S}, t_2, \mathsf{U}, e_1)$

$\pi_{\mathsf{S}}^{t_2} \neq \perp$: **return** $\perp$
if $(\mathsf{U},\mathsf{S}) \in \mathcal{C}$
  $\pi_{\mathsf{S}}^{t_2}.\mathtt{fr} :=$ **false**
  $\mathsf{pk} := \mathsf{D}_1(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, e_1)$
  $(\mathsf{c},\mathsf{k}) \leftarrow \mathsf{Encaps}(\mathsf{pk})$
  $e_2 := \mathsf{E}_2(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, \mathsf{c})$
  $\mathrm{ctxt} := (\mathsf{U},\mathsf{S},e_1,e_2)$
  $\mathsf{SK} := \mathsf{H}(\mathrm{ctxt}, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw}_{\mathsf{U},\mathsf{S}})$
else
  $\pi_{\mathsf{S}}^{t_2}.\mathtt{fr} :=$ **true**
  $\mathsf{pk} := \mathsf{D}_1(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, e_1)$
  $(\mathsf{c},\mathsf{k}) \leftarrow \mathsf{Encaps}(\mathsf{pk})$
  $e_2 := \mathsf{E}_2(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, \mathsf{c})$
  $\mathrm{ctxt} := (\mathsf{U},\mathsf{S},e_1,e_2)$
  $\mathsf{SK} := \mathsf{H}(\mathrm{ctxt}, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw}_{\mathsf{U},\mathsf{S}})$
  if $e_1 \notin \mathcal{L}_1^{\mathsf{U}}$ **and** $\exists \mathsf{pk}$ s.t. $(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, \mathsf{pk}, e_1, \mathrm{enc}) \in \mathcal{L}_1$ // $\mathbf{G}_7$-$\mathbf{G}_{10}$
    $\mathsf{Guess}_{\mathrm{ser}} :=$ **true** // $\mathbf{G}_7$-$\mathbf{G}_{10}$
  else
    $\mathsf{SK} \xleftarrow{\$} \mathcal{SK}$ // $\mathbf{G}_9$-$\mathbf{G}_{10}$
    $\mathsf{c} \xleftarrow{\$} \mathcal{C}, e_2 := \mathsf{E}_2(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, \mathsf{c})$ // $\mathbf{G}_{10}$
$\pi_{\mathsf{S}}^{t_2}.(\mathsf{e}, \mathtt{tr}) := ((\mathsf{c},\mathsf{k},e_2), \mathrm{ctxt})$
$\pi_{\mathsf{S}}^{t_2}.(\mathtt{key}, \mathtt{acc}) := (\mathsf{SK},$ **true**$)$
$\mathcal{L}_2^{\mathsf{S}} := \mathcal{L}_2^{\mathsf{S}} \cup \{e_2\}$ // $\mathbf{G}_7$-$\mathbf{G}_{10}$
return (S, e_2)

Oracle $\textsc{SendInit}(\mathsf{U}, t_1, \mathsf{S})$

if $\pi_{\mathsf{U}}^{t_1} \neq \perp$: **return** $\perp$
$(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KG}(\mathsf{par}), e_1 := \mathsf{E}_1(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, \mathsf{pk})$
$e_1 \xleftarrow{\$} \mathcal{E}_1 \setminus \mathcal{T}_1, \mathcal{T}_1 := \mathcal{T}_1 \cup \{e_1\}$ // $\mathbf{G}_9$-$\mathbf{G}_{10}$
$\mathsf{pk} := \mathsf{D}_1(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, e_1)$ // $\mathbf{G}_9$-$\mathbf{G}_{10}$
Retrieve sk s.t. $(\mathsf{pk},\mathsf{sk}) \in \mathcal{L}_{\mathsf{key}}$ // $\mathbf{G}_9$-$\mathbf{G}_{10}$
$\mathcal{L}_1^{\mathsf{U}} := \mathcal{L}_1^{\mathsf{U}} \cup \{e_1\}$ // $\mathbf{G}_7$-$\mathbf{G}_{10}$
$\pi_{\mathsf{U}}^{t_1} := ((\mathsf{pk},\mathsf{sk},e_1), (\mathsf{U},\mathsf{S},e_1,\perp), \perp, \perp)$
$\pi_{\mathsf{U}}^{t_1}.\mathtt{fr} :=$ **false**
return (U, e_1)

Oracle $\textsc{SendTerInit}(\mathsf{U}, t_1, \mathsf{S}, e_2)$

if $\pi_{\mathsf{U}}^{t_1} = \perp$ **and** $\pi_{\mathsf{U}}^{t_1}.\mathtt{tr} \neq (\mathsf{U},\mathsf{S},*,*)$
  return $\perp$
$(\mathsf{pk},\mathsf{sk},e_1) := \pi_{\mathsf{U}}^{t_1}.\mathsf{e}$
if $\exists t_2$ s.t. $\pi_{\mathsf{S}}^{t_2}.\mathtt{fr} =$ **true**
  and $\pi_{\mathsf{S}}^{t_2}.\mathtt{tr} = (\mathsf{U},\mathsf{S},e_1,e_2)$
  $\pi_{\mathsf{U}}^{t_1}.\mathtt{fr} :=$ **true**, $\mathsf{SK} := \pi_{\mathsf{S}}^{t_2}.\mathtt{key}$
else
  $\mathrm{ctxt} := (\mathsf{U},\mathsf{S},e_1,e_2)$
  if $(\mathsf{U},\mathsf{S}) \notin \mathcal{C}$
    $\pi_{\mathsf{U}}^{t_1}.\mathtt{fr} :=$ **true**
    $\mathsf{c} := \mathsf{D}_2(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, e_2)$, $\mathsf{k} := \mathsf{Decaps}(\mathsf{sk},\mathsf{c})$
    $\mathsf{SK} := \mathsf{H}(\mathrm{ctxt}, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw}_{\mathsf{U},\mathsf{S}})$
    if $e_2 \notin \mathcal{L}_2^{\mathsf{S}}$ **and** $\exists \mathsf{c}$ s.t. $(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, \mathsf{c}, e_2, \mathrm{enc}) \in \mathcal{L}_2$ // $\mathbf{G}_7$-$\mathbf{G}_{10}$
      $\mathsf{Guess}_{\mathrm{user}} :=$ **true** // $\mathbf{G}_7$-$\mathbf{G}_{10}$
    else // $\mathbf{G}_8$-$\mathbf{G}_{10}$
      $\mathsf{SK} \xleftarrow{\$} \mathcal{SK}$ // $\mathbf{G}_8$-$\mathbf{G}_{10}$
  else
    $\pi_{\mathsf{U}}^{t_1}.\mathtt{fr} :=$ **false**
    $\mathsf{c} := \mathsf{D}_2(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, e_2)$, $\mathsf{k} := \mathsf{Decaps}(\mathsf{sk},\mathsf{c})$
    $\mathsf{SK} := \mathsf{H}(\mathrm{ctxt}, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw}_{\mathsf{U},\mathsf{S}})$
$\pi_{\mathsf{U}}^{t_1}.(\mathtt{tr}, \mathtt{key}, \mathtt{acc}) := (\mathrm{ctxt}, \mathsf{SK},$ **true**$)$
return true

Fig. 11. Oracles $\textsc{SendInit}, \textsc{SendResp}$, and $\textsc{SendTerInit}$ in games $\mathbf{G}_6$-$\mathbf{G}_{10}$. For any user U, $\mathcal{L}_1^{\mathsf{U}}$ records all e_1 sent by U. Similarly, $\mathcal{L}_2^{\mathsf{S}}$ records all e_2 sent by server S. All these lists are initialized as $\emptyset$.

These two flags are internal and do not influence the game, and thus **G7** is equivalent to $\mathbf{G}_6$.

$$\Pr\left[\mathbf{G}_6^{\mathcal{A}} \Rightarrow 1\right] = \Pr\left[\mathbf{G}_7^{\mathcal{A}} \Rightarrow 1\right].$$

This step is crucial for our proof. Looking ahead, $\mathcal{A}$ triggered $\mathsf{Guess}_{\mathrm{user}}$ (or $\mathsf{Guess}_{\mathrm{ser}}$, similarly) means that $\mathcal{A}$ queried $\mathsf{E}_1(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, \mathsf{pk})$ for some pk without corrupting $\mathsf{pw}_{\mathsf{U},\mathsf{S}}$. In this case, such pk is controlled by $\mathcal{A}$ (i.e., not output by the security game), and thus we cannot embed challenge public key into such pk when constructing reduction. Such events happen means that the adversary performs a successful online dictionary attack. We delay the analysis of the happening probability of such events.

Game $\mathbf{G}_8$. Fresh user instances that do not have matching session and do not trigger $\mathsf{Guess}_{\mathrm{user}}$ will generate uniformly random session keys. Concretely, when $\mathcal{A}$ queries $\textsc{SendTerInit}(\mathsf{U}, t_1, \mathsf{S}, e_2)$, if $\pi_{\mathsf{U}}^{t_1}$ does not have matching instance,

Reduction $\mathcal{B}_4^{\text{PCO}}(\mathsf{par}, \mathbf{pk}, \mathbf{c})$

01 $\mathsf{cnt}_1 := 0, \mathsf{cnt}_2 := 0, \mathcal{L}_{\text{ct}} := \emptyset$
02 $i^* := \bot, j^* := \bot, \mathsf{k}^* := \bot$
03 **for** $(\mathsf{U}, \mathsf{S}) \in \mathcal{U}$: $\mathsf{pw}_{\mathsf{U},\mathsf{S}} \leftarrow \mathcal{PW}$
04 $\mathcal{C} := \emptyset, \beta \leftarrow \{0, 1\}$
05 $\mathsf{Guess}_{\text{user}} :=$ **false**, $\mathsf{Guess}_{\text{ser}} :=$ **false**
06 $\mathsf{Query}_{\text{send}} :=$ **false**
07 $b' \leftarrow \mathcal{A}^{O,\mathsf{H},\mathsf{IC}_1,\mathsf{IC}_2}(\mathsf{par})$
08 **return** (i^*, j^*, k^*)

Oracle $\text{SENDINIT}(\mathsf{U}, t_1, \mathsf{S})$

09 **if** $\pi_{\mathsf{U}}^{t_1} \neq \bot$: **return** $\bot$
10 $\mathsf{cnt}_1 := \mathsf{cnt}_1 + 1, \mathsf{pk} := \mathbf{pk}[\mathsf{cnt}_1]$
11 $e_1 := \mathsf{E}_1(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, \mathsf{pk}), \mathcal{L}_1^{\mathsf{U}} := \mathcal{L}_1^{\mathsf{U}} \cup \{e_1\}$
12 $\pi_{\mathsf{U}}^{t_1} := ((\mathsf{pk}, \mathsf{cnt}_1, e_1), (\mathsf{U}, \mathsf{S}, e_1, \bot), \bot, \bot)$
13 **return** (U, e_1)

Oracle $\mathsf{D}_2(\mathsf{pw}, e_2)$

14 **if** $\exists(\mathsf{pw}, \mathsf{c}, e_2, *) \in \mathcal{L}_2$: **return** c
15 $\mathsf{cnt}_2 := \mathsf{cnt}_2 + 1, \mathsf{c} := \mathbf{c}[\mathsf{cnt}_2]$
16 $\mathcal{L}_{\text{ct}} := \mathcal{L}_{\text{ct}} \cup \{(\mathsf{c}, \mathsf{cnt}_2)\}$
17 $\mathcal{L}_2 := \mathcal{L}_2 \cup (\mathsf{pw}, \mathsf{c}, e_2, \text{dec})$
18 **return** c

Oracle $\mathsf{H}(\mathsf{U}, \mathsf{S}, e_1, e_2, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw})$

19 $\text{ctxt} := (\mathsf{U}, \mathsf{S}, e_1, e_2)$
20 **if** $\exists i, \mathsf{SK}$ s.t. $(\text{ctxt}, (\mathsf{pk}, i), \mathsf{c}, \mathsf{pw}, \mathsf{SK}) \in \mathcal{L}'_{\text{SK}}$
21 **and** $\text{PCO}(i, \mathsf{c}, \mathsf{k}) = 1$
22 $\mathcal{L}_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, e_1, e_2, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw}] := \mathsf{SK}$
23 **if** $\exists i, j$ s.t. $(\text{ctxt}, (\mathsf{pk}, i), (\mathsf{c}, j)) \in \mathcal{L}_{\text{SK}}$
24 **and** $\text{PCO}(i, \mathsf{c}, \mathsf{k}) = 1$
25 $(i^*, j^*, \mathsf{k}^*) := (i, j, \mathsf{k}), \mathsf{Query}_{\text{send}} :=$ **true**
26 **if** $\mathcal{L}_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, e_1, e_2, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw}] = \bot$
27 $\mathcal{L}_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, e_1, e_2, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw}] := \mathsf{SK} \xleftarrow{\$} \mathcal{SK}$
28 **return** $\mathcal{L}_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, e_1, e_2, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw}]$

Oracle $\text{SENDTERINIT}(\mathsf{U}, t_1, \mathsf{S}, e_2)$

29 **if** $\pi_{\mathsf{U}}^{t_1} = \bot$ **and** $\pi_{\mathsf{U}}^{t_1}.\mathtt{tr} \neq (\mathsf{U}, \mathsf{S}, *, *)$
30 **return** $\bot$
31 $(\mathsf{pk}, i, e_1) := \pi_{\mathsf{U}}^{t_1}.\mathtt{e}$
32 **if** $\exists t_2$ s.t. $\pi_{\mathsf{S}}^{t_2}.\mathtt{fr} =$ **true**
33 **and** $\pi_{\mathsf{S}}^{t_2}.\mathtt{tr} = (\mathsf{U}, \mathsf{S}, e_1, e_2)$
34 $\pi_{\mathsf{U}}^{t_1}.\mathtt{fr} :=$ **true**, $\mathsf{SK} := \pi_{\mathsf{S}}^{t_2}.\mathtt{key}$
35 **else**
36 $\text{ctxt} := (\mathsf{U}, \mathsf{S}, e_1, e_2), \mathsf{c} := \mathsf{D}_2(\mathsf{pw}, e_2)$
37 **if** $(\mathsf{U}, \mathsf{S}) \notin \mathcal{C}$
38 $\pi_{\mathsf{U}}^{t_1}.\mathtt{fr} :=$ **true**
39 $\mathsf{c} := \mathsf{D}_2(\mathsf{pw}, e_2)$
40 **if** $e_2 \notin \mathcal{L}_2^{\mathsf{S}}$ **and** $\exists \mathsf{c}$ s.t. $(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, \mathsf{c}, e_2, \text{enc}) \in \mathcal{L}_2$
41 $\mathsf{Guess}_{\text{user}} :=$ **true**
42 $\mathsf{SK} := \mathsf{Patch}(\text{ctxt}, \mathsf{pk}, i, \mathsf{c})$
43 **else**
44 Retrieve j s.t. $(\mathsf{c}, j) \in \mathcal{L}_{\text{ct}}$
45 $\mathsf{SK} \xleftarrow{\$} \mathcal{SK}$
46 $\mathcal{L}_{\text{SK}} := \mathcal{L}_{\text{SK}} \cup (\text{ctxt}, (\mathsf{pk}, i), (\mathsf{c}, j))$
47 **else**
48 $\pi_{\mathsf{U}}^{t_1}.\mathtt{fr} :=$ **false**
49 $\mathsf{SK} := \mathsf{Patch}(\text{ctxt}, \mathsf{pk}, i, \mathsf{c})$
50 $\pi_{\mathsf{U}}^{t_1}.(\mathtt{tr}, \mathtt{key}, \mathtt{acc}) := (\text{ctxt}, \mathsf{SK},$ **true**$)$
51 **return true**

Procedure $\mathsf{Patch}(\text{ctxt}, \mathsf{pk}, i, \mathsf{c})$

52 $(\mathsf{U}, \mathsf{S}, e_1, e_2) := \text{ctxt}, \mathsf{pw} := \mathsf{pw}_{\mathsf{U},\mathsf{S}}$
53 **if** $\exists \mathsf{k}$ s.t. $\text{PCO}(i, \mathsf{k}, \mathsf{c}) = 1$
54 **and** $\mathcal{L}_{\mathsf{H}}[\text{ctxt}, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw}] \neq \bot$
55 $\mathsf{SK} := \mathcal{L}_{\mathsf{H}}[\text{ctxt}, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw}]$
56 **else**
57 $\mathsf{SK} \xleftarrow{\$} \mathcal{SK}$
58 $\mathcal{L}'_{\text{SK}} := \mathcal{L}'_{\text{SK}} \cup (\text{ctxt}, (\mathsf{pk}, i), \mathsf{c}, \mathsf{pw}, \mathsf{SK})$
59 **return** SK

Fig. 12. Reduction $\mathcal{B}_4$ in bounding the probability difference between $\mathbf{G}_7$ and $\mathbf{G}_8$. Highlighted parts show how $\mathcal{B}_4$ uses PCO and challenge input to simulate $\mathbf{G}_8$. $\mathcal{A}_4$ also uses a procedure Patch to patch H. All other oracles not shown in the figure are the same as in $\mathbf{G}_8$ (cf. Figs. 8, 9 and 11).

(U, S) is uncorrupted, and e_2 does not trigger $\mathsf{Guess}_{\text{user}}$, then we sample the session key uniformly at random and independent of H (cf. Lines 55 ro 56).

Since session keys in $\mathbf{G}_7$ are generated via random oracle H, to distinguish $\mathbf{G}_8$ and $\mathbf{G}_7$, $\mathcal{A}$ needs to query one of the intended hash inputs of such random session keys. Let $\mathsf{Query}_{\text{send}}$ be such querying event. To bound the happening probability of $\mathsf{Query}_{\text{send}}$, we construct an reduction $\mathcal{B}_4$ with $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B}_4)$ in Fig. 12 which attacks OW-rPCA security of KEM. $\mathcal{B}_4$ works as follows:

1. On input a OW-rPCA challenge $(\mathsf{par}, \mathbf{pk}, \mathbf{c})$, $\mathcal{B}_4$ embeds public keys in $\mathbf{pk}$ into queries to SENDINIT (cf. Line 02) and embeds challenge ciphertexts in D_2 (cf. Line 15). Counter cnt_1 and cnt_2 are used to record the indexes of embedded public keys and ciphertexts, respectively.
2. Since $\mathcal{B}_4$ does not have secret keys of challenge public keys (cf. Line 02), it cannot decrypt KEM ciphertexts and thus cannot directly compute session

keys of user instances or determine whether $\mathcal{A}$ has queried the hash input of such session keys (even if these keys are not fresh). To deal with it, we use RO patching technique to make the simulation consistent.
Concretely, we define a procedure Patch which uses PCO oracle to determine if $\mathcal{A}$ has queried the intended hash input of the session key of some specific user instances. If so, it returns the recorded session key. Otherwise, it samples a random session key, records this session key in $\mathcal{L}'_{\text{SK}}$, and returns it. Later, if $\mathcal{A}$'s RO query matches a recorded session key, then $\mathcal{B}_4$ patches the RO and returns this key (cf. Lines 20 to 22).
When $\mathcal{A}$ queries SENDTERINIT$(\mathsf{U}, t_1, \mathsf{S}, e_2)$, where $\pi_{\mathsf{U}}^{t_1}$ does not have fresh matching instance and either e_2 triggers $\mathsf{Guess}_{\text{user}}$ or (U, S) is corrupted, $\mathcal{B}_4$ uses the procedure to compute the session key (cf. Lines 42 and 49).
3. When $\mathcal{A}$ queries SENDTERINIT$(\mathsf{U}, t_1, \mathsf{S}, e_2)$, if $\pi_{\mathsf{U}}^{t_1}$ does not have fresh matching instance, (U, S) is corrupted, and e_2 does not trigger $\mathsf{Guess}_{\text{user}}$, then e_2 is not generated by querying $\mathsf{E}_2(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, e_2)$, which means that $\mathsf{c} = \mathsf{D}_2(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, e_2)$ is one of the embedded ciphertext (cf. Line 15). $\mathcal{B}_4$ records such query in $\mathcal{L}_{\text{SK}}$ (cf. Line 46) to determine whether $\mathsf{Query}_{\text{send}}$ happens.
When $\mathcal{A}$ queried $\mathsf{H}(\mathsf{U}, \mathsf{S}, e_1, e_2, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw}_{\mathsf{U},\mathsf{S}})$, if this query match one record in $\mathcal{L}_{\text{SK}}$ and k is the decapsulated key of a embedded challenge ciphertext c (cf. Line 23), then this RO query is the intended hash input of one of the session keys recorded in Line 46. In this case, $\mathsf{Query}_{\text{send}}$ will be triggered, and $\mathcal{B}_4$ will use (i^*, j^*, k^*) to record the OW solution of c (cf. Line 25).

Since $\mathcal{A}$'s numbers of queries to Init and D_2 are S and q_2, respectively, $\mathcal{B}_4$ needs at most S challenge public keys and $(q_2 + S)$ challenge ciphertexts per public keys during the simulation. If $\mathsf{Query}_{\text{send}}$ happens, then $\mathcal{B}_4$ finds the OW solution of one of the challenge ciphertexts. Therefore, we have

$$\left|\Pr\left[\mathbf{G}_7^{\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\mathbf{G}_8^{\mathcal{A}} \Rightarrow 1\right]\right| \leq \Pr\left[\mathsf{Query}_{\text{send}}\right] \leq \mathsf{Adv}_{\mathsf{KEM}}^{(S, q_2+S)\text{-}\mathsf{OW}\text{-}\mathsf{rPCA}}(\mathcal{B}_4)$$

Game $\mathbf{G}_9$. We change SENDINIT and SENDRESP.

1. In SENDINIT, instead of generating $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}$ and $e_1 := \mathsf{E}_1(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, \mathsf{pk})$, we firstly sample e_1 uniformly at random and then generate $(\mathsf{pk}, \mathsf{sk})$ by querying $\mathsf{D}_1(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, e_1)$ (cf. Lines 34 to 36).
2. Fresh server instances that do not trigger $\mathsf{Guess}_{\text{ser}}$ will generate uniformly random session keys. Concretely, when $\mathcal{A}$ queries SENDRESP$(\mathsf{S}, t_2, \mathsf{U}, e_1)$, if (U, S) is uncorrupted and e_1 does not trigger $\mathsf{Guess}_{\text{ser}}$, then we sample the session key uniformly at random and independent of H (cf. Lines 25 to 26).

Similar to our argument in bounding $\mathbf{G}_7$ and $\mathbf{G}_8$, to distinguish $\mathbf{G}_8$ and $\mathbf{G}_9$, $\mathcal{A}$ needs to query one of the intended hash inputs of such random session keys. Let $\mathsf{Query}_{\text{resp}}$ be such querying event. We construct an reduction $\mathcal{B}_5$ with $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B}_5)$ in Fig. 12 to bound the happening probability of $\mathsf{Query}_{\text{resp}}$. $\mathcal{B}_5$ attacks $\mathsf{OW}\text{-}\mathsf{PCA}$ security of KEM and works as follows:

1. On input a $\mathsf{OW}\text{-}\mathsf{PCA}$ challenge $(\mathsf{par}, \mathbf{pk}, \mathbf{c})$, $\mathcal{B}_5$ embeds challenge public keys $\mathbf{pk}$ into queries to D_1 (cf. Line 31). By Lines 34 to 36, public keys generated

Reduction $\mathcal{B}_5(\mathsf{par}, \mathbf{pk}, \mathbf{c})$

01 $\mathsf{cnt}_1 := 0, i^* := \bot, j^* := \bot, \mathsf{k}^* := \bot$
02 **for** $(\mathsf{U}, \mathsf{S}) \in \mathcal{U}$
03 $\quad \mathsf{pw}_{\mathsf{U},\mathsf{S}} \leftarrow \mathcal{PW}, \mathcal{L}_1^{\mathsf{U}} := \emptyset, \mathcal{L}_2^{\mathsf{S}} := \emptyset$
04 $\mathcal{C} := \emptyset, \beta \leftarrow \{0,1\}$
05 $\mathsf{Guess}_{\mathrm{user}} :=$ **false**, $\mathsf{Guess}_{\mathrm{ser}} :=$ **false**
06 $\mathsf{Query}_{\mathrm{resp}} :=$ **false**
07 $b' \leftarrow \mathcal{A}^{O,\mathsf{H},\mathsf{IC}_1,\mathsf{IC}_2}(\mathsf{par})$
08 **return** (i^*, j^*, k^*)

Oracle $\textsc{SendTerInit}(\mathsf{U}, t_1, \mathsf{S}, e_2)$

09 **if** $\pi_{\mathsf{U}}^{t_1} = \bot$ **and** $\pi_{\mathsf{U}}^{t_1}.\mathbf{tr} \neq (\mathsf{U}, \mathsf{S}, *, *)$
10 $\quad$ **return** $\bot$
11 $(\mathsf{pk}, i, e_1) := \pi_{\mathsf{U}}^{t_1}.\mathsf{e}, \mathsf{c} := \mathsf{D}_2(\mathsf{pw}, e_2)$
12 **if** $\exists t_2$ s.t. $\pi_{\mathsf{S}}^{t_2}.\mathbf{fr} =$ **true**
13 $\quad$ **and** $\pi_{\mathsf{S}}^{t_2}.\mathbf{tr} = (\mathsf{U}, \mathsf{S}, e_1, e_2)$
14 $\quad \pi_{\mathsf{U}}^{t_1}.\mathbf{fr} :=$ **true**, $\mathsf{SK} := \pi_{\mathsf{S}}^{t_2}.\mathbf{key}$
15 **else**
16 $\quad \mathrm{ctxt} := (\mathsf{U}, \mathsf{S}, e_1, e_2)$
17 $\quad$ **if** $(\mathsf{U}, \mathsf{S}) \notin \mathcal{C}$
18 $\quad\quad \pi_{\mathsf{U}}^{t_1}.\mathbf{fr} :=$ **true**
19 $\quad\quad$ **if** $e_2 \notin \mathcal{L}_2^{\mathsf{S}}$ **and** $\exists \mathsf{c}$ s.t. $(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, \mathsf{c}, e_2, \mathrm{enc}) \in \mathcal{L}_2$
20 $\quad\quad\quad \mathsf{Guess}_{\mathrm{user}} :=$ **true**
21 $\quad\quad\quad \mathsf{SK} := \mathsf{Patch}(\mathrm{ctxt}, \mathsf{pk}, i, \mathsf{c})$
22 $\quad\quad$ **else** $\mathsf{SK} \xleftarrow{\$} \mathcal{SK}$
23 $\quad$ **else**
24 $\quad\quad \pi_{\mathsf{U}}^{t_1}.\mathbf{fr} :=$ **false**
25 $\quad\quad \mathsf{SK} := \mathsf{Patch}(\mathrm{ctxt}, \mathsf{pk}, i, \mathsf{c})$
26 $\pi_{\mathsf{U}}^{t_1}.(\mathbf{tr}, \mathbf{key}, \mathbf{acc}) := (\mathrm{ctxt}, \mathsf{SK}, \mathbf{true})$
27 **return true**

Oracle $\mathsf{D}_1(\mathsf{pw}, e_1)$

28 **if** $\exists (\mathsf{pw}, \mathsf{pk}, e_1, *) \in \mathcal{L}_1$
29 $\quad$ **return** c
30 $\mathsf{cnt}_2[\mathsf{cnt}_1] := 0, \mathsf{cnt}_1 := \mathsf{cnt}_1 + 1$
31 $\mathsf{pk} := \mathbf{pk}[\mathsf{cnt}_1], \mathcal{L}_{\mathsf{key}} := \mathcal{L}_{\mathsf{key}} \cup \{(\mathsf{pk}, \mathsf{cnt}_1)\}$
32 $\mathcal{L}_2 := \mathcal{L}_2 \cup (\mathsf{pw}, \mathsf{pk}, e_1, \mathrm{dec})$
33 **return** c

Oracle $\textsc{SendResp}(\mathsf{S}, t_2, \mathsf{U}, e_1)$

34 $\pi_{\mathsf{S}}^{t_2} \neq \bot$: **return** $\bot$
35 $\mathsf{pk} := \mathsf{D}_1(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, e_1)$
36 **if** $(\mathsf{U}, \mathsf{S}) \in \mathcal{C}$
37 $\quad \pi_{\mathsf{S}}^{t_2}.\mathbf{fr} :=$ **false**
38 $\quad (\mathsf{c}, \mathsf{k}) \leftarrow \mathsf{Encaps}(\mathsf{pk}), e_2 := \mathsf{E}_2(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, \mathsf{c})$
39 $\quad \mathrm{ctxt} := (\mathsf{U}, \mathsf{S}, e_1, e_2)$
40 $\quad \mathsf{SK} := \mathsf{H}(\mathrm{ctxt}, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw}_{\mathsf{U},\mathsf{S}})$
41 **else**
42 $\quad \pi_{\mathsf{S}}^{t_2}.\mathbf{fr} :=$ **true**
43 $\quad$ **if** $e_1 \notin \mathcal{L}_1^{\mathsf{U}}$ **and** $\exists \mathsf{pk}$ s.t. $(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, \mathsf{pk}, e_1, \mathrm{enc}) \in \mathcal{L}_1$
44 $\quad\quad \mathsf{Guess}_{\mathrm{ser}} :=$ **true**
45 $\quad\quad (\mathsf{c}, \mathsf{k}) \leftarrow \mathsf{Encaps}(\mathsf{pk}), e_2 := \mathsf{E}_2(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, \mathsf{c})$
46 $\quad\quad \mathsf{SK} := \mathsf{H}(\mathrm{ctxt}, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw}_{\mathsf{U},\mathsf{S}})$
47 $\quad$ **else**
48 $\quad\quad$ Retrieve i s.t. $(\mathsf{pk}, i) \in \mathcal{L}_{\mathsf{key}}$
49 $\quad\quad \mathsf{cnt}_2[i] := \mathsf{cnt}_2[i] + 1, j := \mathsf{cnt}_2[i]$
50 $\quad\quad \mathsf{c} := \mathbf{c}[i, j],\ e_2 := \mathsf{E}_2(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, \mathsf{c})$
51 $\quad\quad \mathcal{L}_{\mathrm{SK}} := \mathcal{L}_{\mathrm{SK}} \cup \{(\mathrm{ctxt}, (\mathsf{pk}, i), (\mathsf{c}, j))\}$
52 $\quad\quad \mathsf{SK} \xleftarrow{\$} \mathcal{SK}$
53 $\mathcal{L}_2^{\mathsf{S}} := \mathcal{L}_2^{\mathsf{S}} \cup \{e_2\}$
54 $\pi_{\mathsf{S}}^{t_2} := ((\mathsf{c}, \mathsf{k}, e_2), \mathrm{ctxt}, \mathsf{SK}, \mathbf{true})$
55 **return** (S, e_2)

Oracle $\mathsf{H}((\mathsf{U}, \mathsf{S}, e_1, e_2), \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw})$

56 $\mathrm{ctxt} := (\mathsf{U}, \mathsf{S}, e_1, e_2)$
57 **if** $\exists i, \mathsf{SK}$ s.t. $(\mathrm{ctxt}, (\mathsf{pk}, i), \mathsf{c}, \mathsf{pw}, \mathsf{SK}) \in \mathcal{L}'_{\mathrm{SK}}$ **and** $\textsc{Pco}(i, \mathsf{c}, \mathsf{k}) = 1$
58 $\quad \mathcal{L}_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, e_1, e_2, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw}] := \mathsf{SK}$
59 **if** $\exists i, j$ s.t. $(\mathrm{ctxt}, (\mathsf{pk}, i), (\mathsf{c}, j)) \in \mathcal{L}_{\mathrm{SK}}$ **and** $\textsc{Pco}(i, \mathsf{c}, \mathsf{k}) = 1$
60 $\quad (i^*, j^*, \mathsf{k}^*) := (i, j, \mathsf{k}), \mathsf{Query}_{\mathrm{resp}} :=$ **true**
61 **if** $\mathcal{L}_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, e_1, e_2, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw}] = \bot$
62 $\quad \mathcal{L}_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, e_1, e_2, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw}] := \mathsf{SK} \xleftarrow{\$} \mathcal{SK}$
63 **return** $\mathcal{L}_{\mathsf{H}}[\mathsf{U}, \mathsf{S}, e_1, e_2, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw}]$

Fig. 13. Reduction $\mathcal{B}_5$ in bounding the probability difference between $\mathbf{G}_8$ and $\mathbf{G}_9$. Highlighted parts show how $\mathcal{B}_5$ uses Pco and challenge input to simulate $\mathbf{G}_9$. All other oracles not shown in the figure are the same as in $\mathbf{G}_8$ (cf. Figs. 8, 9 and 11). Procedure Patch is the same as the one shown in Fig. 12.

in SendInit are also from **pk**. Similar to $\mathcal{B}_4$, $\mathcal{B}_5$ uses the Patch procedure in Fig. 12 to compute the session keys of user instances. Counter cnt_1 and vector of counters cnt_2 are used to record the indexes of embedded public keys and ciphertexts, respectively.

2. When $\mathcal{A}$ queries $\textsc{SendResp}(\mathsf{S}, t_2, \mathsf{U}, e_1)$, if $\pi_{\mathsf{S}}^{t_2}$ is fresh (which means that (U, S) is uncorrupted) and e_1 does not trigger $\mathsf{Guess}_{\mathrm{ser}}$, then by our definition of $\mathsf{Guess}_{\mathrm{ser}}$, e_1 is not generated by querying $\mathsf{E}_1(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, \mathsf{pk})$. This means that $\mathsf{pk} = \mathsf{D}_1(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, e_1)$ is one of the embedded public key (cf. Line 31). In this case, $\mathcal{B}_5$ embeds one challenge ciphertext with respect to pk (cf. Line 50)

and records such query in $\mathcal{L}_{\mathrm{SK}}$ (cf. Line 51) to determine whether $\mathsf{Query}_{\mathrm{resp}}$ happens.
When $\mathcal{A}$ queried $\mathsf{H}(\mathsf{U}, \mathsf{S}, e_1, e_2, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw}_{\mathsf{U},\mathsf{S}})$, if this query match one record in $\mathcal{L}_{\mathrm{SK}}$ and k is the decapsulated key of a embedded challenge ciphertext c (cf. Line 59), then this RO query is the intended hash input of one of the session keys recorded in Line 51. In this case, $\mathsf{Query}_{\mathrm{resp}}$ will be triggered, and $\mathcal{B}_5$ will use (i^*, j^*, k^*) to record the OW solution of the embedded challenge ciphertext c (cf. Line 60).

Since $\mathcal{A}$'s numbers of queries to (SENDINIT, SENDRESP) and D_2 are S and q_2 respectively, $\mathcal{B}_5$ needs at most $S + q_2$ challenge public keys and S challenge ciphertexts per public keys during the simulation. If $\mathsf{Query}_{\mathrm{resp}}$ happens, then $\mathcal{B}_5$ finds the OW solution of one of challenge ciphertexts in $\mathbf{c}$. Therefore, we have

$$\left|\Pr\left[\mathbf{G}_8^{\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\mathbf{G}_9^{\mathcal{A}} \Rightarrow 1\right]\right| \leq \Pr\left[\mathsf{Query}_{\mathrm{resp}}\right] \leq \mathsf{Adv}_{\mathsf{KEM}}^{(S+q_2,S)\text{-}\mathsf{OW}\text{-}\mathsf{PCA}}(\mathcal{B}_5)$$

<u>**Game $\mathbf{G}_{10}$.**</u> We sample KEM ciphertext uniformly at random for server instances that are fresh and do not trigger $\mathsf{Query}_{\mathrm{resp}}$ (cf. Line 27). Similar to the argument of bounding $\mathbf{G}_3$ and $\mathbf{G}_4$ (cf. Lemma 3), We can use the ciphertext anonymity of KEM to upper bound the probability difference between $\mathbf{G}_9$ and $\mathbf{G}_{10}$. The bound is given in Lemma 4. We continue the proof of Theorem 1 and postpone the proof of Lemma 4 to our full version [28].

Lemma 4. *With notations and assumptions from $\mathbf{G}_9$ and $\mathbf{G}_{10}$ in the proof of Theorem 1, there is an adversary $\mathcal{B}_6$ with $\mathbf{T}(\mathcal{B}_6) \approx \mathbf{T}(\mathcal{A})$ and*

$$\left|\Pr\left[\mathbf{G}_9^{\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\mathbf{G}_{10}^{\mathcal{A}} \Rightarrow 1\right]\right| \leq \mathsf{Adv}_{\mathsf{KEM}}^{(S+q_1,S)\text{-}\mathsf{ANO}}(\mathcal{B}_6)$$

In game transition $\mathbf{G}_{10}$-$\mathbf{G}_{12}$ (shown in Fig. 14), we bound the happening probabilities of $\mathsf{Guess}_{\mathrm{ser}}$ and $\mathsf{Guess}_{\mathrm{user}}$.

<u>**Game $\mathbf{G}_{11}$.**</u> We do not use passwords to simulate the protocol messages of fresh instances that do not trigger $\mathsf{Guess}_{\mathrm{ser}}$ and $\mathsf{Guess}_{\mathrm{user}}$. Concretely, we change SENDINIT, SENDRESP, and SENDTERINIT as follows:

- In SENDRESP, if the server instance $\pi_{\mathsf{S}}^{t_2}$ is fresh and does not trigger $\mathsf{Guess}_{\mathrm{ser}}$, then we sample e_2 uniformly at random and without using $\mathsf{pw}_{\mathsf{U},\mathsf{S}}$ and c (cf. Lines 33 to 34). Moreover, we only store e_2 as the ephemeral secret of $\pi_{\mathsf{S}}^{t_2}$ (cf. Line 41). These changes are conceptual since we do not need c to compute the session key and if $\mathcal{A}$ queries $\mathsf{D}_2(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, e_2)$ later, then we will return random c (which are the same as in $\mathbf{G}_{10}$).
- Similarly, in SENDINIT, we generate e_1 uniformly at random and without using $\mathsf{pw}_{\mathsf{U},\mathsf{S}}$ and pk (cf. Lines 49 to 52) and only store e_1 as the ephemeral secret of $\pi_{\mathsf{U}}^{t_1}$ (cf. Lines 52 to 53 and Line 59). Later, if $\mathcal{A}$ corrupts (U, S) and queries SENDTERINIT to finish the user instance $\pi_{\mathsf{U}}^{t_1}$, we retrieve necessary information to compute the session key (cf. Lines 82 to 83). These changes

Game $\mathbf{G}_{10}$-$\mathbf{G}_{12}$

$\mathsf{par} \leftarrow \mathsf{Setup}$
$\mathbf{for}\ (\mathsf{U},\mathsf{S}) \in \mathcal{U}$: $\mathsf{pw}_{\mathsf{U},\mathsf{S}} \leftarrow \mathcal{PW}$ // $\mathbf{G}_{10}$-$\mathbf{G}_{11}$
$\mathcal{C} := \emptyset, \beta \leftarrow \{0,1\}$
$\mathsf{Guess}_{\mathrm{user}} := \mathbf{false}, \mathsf{Guess}_{\mathrm{ser}} := \mathbf{false}$
$b' \leftarrow \mathcal{A}^{\mathcal{O},\mathsf{H},\mathsf{IC}_1,\mathsf{IC}_2}(\mathsf{par})$
$\mathbf{for}\ (\mathsf{U},\mathsf{S}) \in \mathcal{U} \times \mathcal{S}$ // $\mathbf{G}_{12}$
  $\mathbf{if}\ (\mathsf{U},\mathsf{S}) \notin \mathcal{C}$: $\mathsf{pw}_{\mathsf{U},\mathsf{S}} \leftarrow \mathcal{PW}$ // $\mathbf{G}_{12}$
  $\mathbf{if}\ \exists \mathsf{S}'$ s.t. $\mathsf{pw}_{\mathsf{U},\mathsf{S}'} \in \mathcal{L}_{\mathrm{pw}}$ // $\mathbf{G}_{12}$
    $\mathsf{Guess}_{\mathrm{user}} := \mathbf{true}$ // $\mathbf{G}_{12}$
  $\mathbf{if}\ \exists \mathsf{U}'$ s.t. $\mathsf{pw}_{\mathsf{U}',\mathsf{S}} \in \mathcal{L}_{\mathrm{pw}}$ // $\mathbf{G}_{12}$
    $\mathsf{Guess}_{\mathrm{ser}} := \mathbf{true}$ // $\mathbf{G}_{12}$
$\mathbf{return}\ \beta == b'$

Oracle $\textsc{Corrupt}(\mathsf{U},\mathsf{S})$

$\mathbf{if}\ (\mathsf{U},\mathsf{S}) \in \mathcal{C}$: $\mathbf{return}\ \perp$
$\mathcal{C} := \mathcal{C} \cup \{(\mathsf{U},\mathsf{S})\}$
$\mathsf{pw}_{\mathsf{U},\mathsf{S}} \leftarrow \mathcal{PW}$ // $\mathbf{G}_{12}$
$\mathbf{return}\ \mathsf{pw}_{\mathsf{U},\mathsf{S}}$

Oracle $\textsc{SendResp}(\mathsf{S}, t_2, \mathsf{U}, e_1)$

$\pi_{\mathsf{S}}^{t_2} \neq \perp$: $\mathbf{return}\ \perp$
$\mathbf{if}\ (\mathsf{U},\mathsf{S}) \in \mathcal{C}$
  $\pi_{\mathsf{S}}^{t_2}.\mathtt{fr} := \mathbf{false}$
  $\mathsf{pk} := \mathsf{D}_1(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, e_1), (\mathsf{c},\mathsf{k}) \leftarrow \mathsf{Encaps}(\mathsf{pk})$
  $e_2 := \mathsf{E}_2(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, \mathsf{c}), \mathrm{ctxt} := (\mathsf{U},\mathsf{S},e_1,e_2)$
  $\mathsf{SK} := \mathsf{H}(\mathrm{ctxt}, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw}_{\mathsf{U},\mathsf{S}})$
$\mathbf{else}$
  $\pi_{\mathsf{S}}^{t_2}.\mathtt{fr} := \mathbf{true}$
  $\mathbf{if}\ e_1 \notin \mathcal{L}_1^{\mathsf{U}}\ \mathbf{and}\ \exists \mathsf{pk}$ s.t.
    $(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, \mathsf{pk}, e_1, \mathrm{enc}) \in \mathcal{L}_1$ // $\mathbf{G}_{10}$-$\mathbf{G}_{11}$
    $\mathsf{Guess}_{\mathrm{ser}} := \mathbf{true}$ // $\mathbf{G}_{10}$-$\mathbf{G}_{11}$
    $\mathsf{pk} := \mathsf{D}_1(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, e_1)$ // $\mathbf{G}_{10}$-$\mathbf{G}_{11}$
    $(\mathsf{c},\mathsf{k}) \leftarrow \mathsf{Encaps}(\mathsf{pk})$ // $\mathbf{G}_{10}$-$\mathbf{G}_{11}$
    $e_2 := \mathsf{E}_2(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, \mathsf{c})$ // $\mathbf{G}_{10}$-$\mathbf{G}_{11}$
    $\mathrm{ctxt} := (\mathsf{U},\mathsf{S},e_1,e_2)$ // $\mathbf{G}_{10}$-$\mathbf{G}_{11}$
    $\mathsf{SK} := \mathsf{H}(\mathrm{ctxt}, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw}_{\mathsf{U},\mathsf{S}})$ // $\mathbf{G}_{10}$-$\mathbf{G}_{11}$
  $\mathbf{else}$ // $\mathbf{G}_{10}$-$\mathbf{G}_{11}$
    $\mathsf{c} \leftarrow \mathcal{C}, e_2 := \mathsf{E}_2(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, \mathsf{c})$ // $\mathbf{G}_{10}$
    $e_2 \xleftarrow{\$} \mathcal{E}_2 \backslash \mathcal{T}_2, \mathcal{T}_2 := \mathcal{T}_2 \cup \{e_2\}$ // $\mathbf{G}_{11}$
    $\mathsf{SK} \xleftarrow{\$} \mathcal{SK}$ // $\mathbf{G}_{10}$-$\mathbf{G}_{11}$
  $\mathbf{if}\ e_1 \notin \mathcal{L}_1^{\mathsf{U}}$ // $\mathbf{G}_{12}$
    $\mathbf{for}\ (\mathsf{pw},\mathsf{pk})$ s.t.
      $(\mathsf{pw}, \mathsf{pk}, e_1, \mathrm{enc}) \in \mathcal{L}_1$ // $\mathbf{G}_{12}$
      $\mathcal{L}_{\mathrm{pw}} := \mathcal{L}_{\mathrm{pw}} \cup \{\mathsf{pw}\}$ // $\mathbf{G}_{12}$
  $e_2 \xleftarrow{\$} \mathcal{E}_2 \backslash \mathcal{T}_2, \mathcal{T}_2 := \mathcal{T}_2 \cup \{e_2\}$ // $\mathbf{G}_{12}$
  $\mathsf{SK} \xleftarrow{\$} \mathcal{SK}$ // $\mathbf{G}_{12}$
$\pi_{\mathsf{S}}^{t_2}.(\mathtt{e},\mathtt{tr}) := ((\mathsf{c},\mathsf{k},e_2), \mathrm{ctxt})$ // $\mathbf{G}_{10}$
$\pi_{\mathsf{S}}^{t_2}.(\mathtt{e},\mathtt{tr}) := ((\perp,\perp,e_2), \mathrm{ctxt})$ // $\mathbf{G}_{11}$-$\mathbf{G}_{12}$
$\pi_{\mathsf{S}}^{t_2}.(\mathtt{key},\mathtt{acc}) := (\mathsf{SK}, \mathbf{true})$
$\mathcal{L}_2^{\mathsf{S}} := \mathcal{L}_2^{\mathsf{S}} \cup \{e_2\}$
$\mathbf{return}\ (\mathsf{S}, e_2)$

Oracle $\textsc{SendInit}(\mathsf{U}, t_1, \mathsf{S})$

$\mathbf{if}\ \pi_{\mathsf{U}}^{t_1} \neq \perp$: $\mathbf{return}\ \perp$
$e_1 \xleftarrow{\$} \mathcal{E}_1 \backslash \mathcal{T}_1, \mathcal{L}_1 := \mathcal{L}_1 \cup \{e_1\}$
$\mathcal{L}_1^{\mathsf{U}} := \mathcal{L}_1^{\mathsf{U}} \cup \{e_1\}$
$\mathsf{pk} := \mathsf{D}_1(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, e_1)$ // $\mathbf{G}_{10}$
Retrieve sk s.t. $(\mathsf{pk},\mathsf{sk}) \in \mathcal{L}_{\mathsf{key}}$ // $\mathbf{G}_{10}$
$\pi_{\mathsf{U}}^{t_1} := ((\mathsf{pk}, \mathsf{sk}, e_1),$
  $(\mathsf{U},\mathsf{S},e_1,\perp)), \perp, \perp)$ // $\mathbf{G}_{10}$
$\pi_{\mathsf{U}}^{t_1}.\mathtt{e} := (\perp,\perp,e_1)$ // $\mathbf{G}_{11}$-$\mathbf{G}_{12}$
$\pi_{\mathsf{U}}^{t_1}.\mathtt{tr} := (\mathsf{U},\mathsf{S},e_1,\perp)$ // $\mathbf{G}_{11}$-$\mathbf{G}_{12}$
$\pi_{\mathsf{U}}^{t_1}.\mathtt{fr} := \mathbf{false}$
$\mathbf{return}\ (\mathsf{U}, e_1)$

Oracle $\textsc{SendTerInit}(\mathsf{U}, t_1, \mathsf{S}, e_2)$

$\mathbf{if}\ \pi_{\mathsf{U}}^{t_1} = \perp\ \mathbf{and}\ \pi_{\mathsf{U}}^{t_1}.\mathtt{tr} \neq (\mathsf{U},\mathsf{S},*,*)$
  $\mathbf{return}\ \perp$
$(\mathsf{pk},\mathsf{sk},e_1) := \pi_{\mathsf{U}}^{t_1}.\mathtt{e}$ // $\mathbf{G}_{10}$
$(\perp,\perp,e_1) := \pi_{\mathsf{U}}^{t_1}.\mathtt{e}$ // $\mathbf{G}_{11}$
$\mathbf{if}\ \exists t_2$ s.t. $\pi_{\mathsf{S}}^{t_2}.\mathtt{fr} = \mathbf{true}$
  $\mathbf{and}\ \pi_{\mathsf{S}}^{t_2}.\mathtt{tr} = (\mathsf{U},\mathsf{S},e_1,e_2)$
  $\pi_{\mathsf{U}}^{t_1}.\mathtt{fr} := \mathbf{true}, \mathsf{SK} := \pi_{\mathsf{S}}^{t_2}.\mathtt{key}$
$\mathbf{else}$
  $\mathrm{ctxt} := (\mathsf{U},\mathsf{S},e_1,e_2)$
  $\mathbf{if}\ (\mathsf{U},\mathsf{S}) \notin \mathcal{C}$
    $\pi_{\mathsf{U}}^{t_1}.\mathtt{fr} := \mathbf{true}$
    $\mathbf{if}\ e_2 \notin \mathcal{L}_2^{\mathsf{S}}\ \mathbf{and}\ \exists \mathsf{c}$ s.t.
      $(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, \mathsf{c}, e_2, \mathrm{enc}) \in \mathcal{L}_2$ // $\mathbf{G}_{10}$-$\mathbf{G}_{11}$
      $\mathsf{pk} := \mathsf{D}_1(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, e_1)$ // $\mathbf{G}_{11}$
      Retrieve sk s.t.
        $(\mathsf{pk},\mathsf{sk}) \in \mathcal{L}_{\mathsf{key}}$ // $\mathbf{G}_{11}$
      $\mathsf{Guess}_{\mathrm{user}} := \mathbf{true}$ // $\mathbf{G}_{10}$-$\mathbf{G}_{11}$
      $\mathsf{c} := \mathsf{D}_2(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, e_2)$ // $\mathbf{G}_{10}$-$\mathbf{G}_{11}$
      $\mathsf{k} := \mathsf{Decaps}(\mathsf{sk}, \mathsf{c})$ // $\mathbf{G}_{10}$-$\mathbf{G}_{11}$
      $\mathsf{SK} := \mathsf{H}(\mathrm{ctxt}, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw}_{\mathsf{U},\mathsf{S}})$ // $\mathbf{G}_{10}$-$\mathbf{G}_{11}$
    $\mathbf{else}$ // $\mathbf{G}_{10}$-$\mathbf{G}_{11}$
      $\mathsf{SK} \xleftarrow{\$} \mathcal{SK}$ // $\mathbf{G}_{10}$-$\mathbf{G}_{11}$
    $\mathbf{if}\ e_2 \notin \mathcal{L}_2^{\mathsf{S}}$
      $\mathbf{for}\ (\mathsf{pw},\mathsf{c})$ s.t.
        $(\mathsf{pw}, \mathsf{c}, e_2, \mathrm{enc}) \in \mathcal{L}_2$ // $\mathbf{G}_{12}$
        $\mathcal{L}_{\mathrm{pw}} := \mathcal{L}_{\mathrm{pw}} \cup \{\mathsf{pw}\}$ // $\mathbf{G}_{12}$
    $\mathsf{SK} \xleftarrow{\$} \mathcal{SK}$ // $\mathbf{G}_{12}$
  $\mathbf{else}$
    $\pi_{\mathsf{U}}^{t_1}.\mathtt{fr} := \mathbf{false}$
    $\mathsf{pk} := \mathsf{D}_1(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, e_1)$ // $\mathbf{G}_{11}$-$\mathbf{G}_{12}$
    Retrieve sk s.t.
      $(\mathsf{pk},\mathsf{sk}) \in \mathcal{L}_{\mathsf{key}}$ // $\mathbf{G}_{11}$-$\mathbf{G}_{12}$
    $\mathsf{c} := \mathsf{D}_2(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, e_2)$
    $\mathsf{k} := \mathsf{Decaps}(\mathsf{sk}, \mathsf{c})$
    $\mathsf{SK} := \mathsf{H}(\mathrm{ctxt}, \mathsf{pk}, \mathsf{c}, \mathsf{k}, \mathsf{pw}_{\mathsf{U},\mathsf{S}})$
$\pi_{\mathsf{U}}^{t_1}.(\mathtt{tr},\mathtt{key},\mathtt{acc}) := (\mathrm{ctxt}, \mathsf{SK}, \mathbf{true})$
$\mathbf{return\ true}$

Fig. 14. Oracles $\textsc{SendInit}$, $\textsc{SendResp}$, and $\textsc{SendTerInit}$ in games $\mathbf{G}_{10}$-$\mathbf{G}_{12}$.

are also conceptual, since session keys of such instances are independently and uniformly random. We have

$$\Pr\left[\mathbf{G}_{10}^{\mathcal{A}} \Rightarrow 1\right] = \Pr\left[\mathbf{G}_{11}^{\mathcal{A}} \Rightarrow 1\right]$$

Game $\mathbf{G}_{12}$. We postpone the generation of passwords and the determination of whether $\mathsf{Guess}_{\text{user}}$ or $\mathsf{Guess}_{\text{ser}}$ happen. For simplicity, we define event GUESS as $\mathsf{Guess}_{\text{user}} \vee \mathsf{Guess}_{\text{ser}}$.

1. We generate passwords as late as possible. passwords are generated only when $\mathcal{A}$ issues Corrupt queries or after $\mathcal{A}$ ends with output b' (cf. Lines 06, 07 to 15).
2. Since the passwords of uncorrupted parties do not exist before $\mathcal{A}$ terminates, we cannot determine whether GUESS happens when $\mathcal{A}$ is running. To deal with it, we postpone such determination. When $\mathcal{A}$ issues SendResp or SendTerInit queries, we records all potential passwords that may match the actual password of the specific user-server pair (cf. Lines 37 to 38 and Lines 76 to 78). After $\mathcal{A}$ outputs b', the passwords of uncorrupted user-server pairs are generated, and then we use these passwords to determine whether $\mathsf{Guess}_{\text{user}}$ or $\mathsf{Guess}_{\text{ser}}$ happen (cf. Lines 06 to 11).
3. Now all fresh instances will accept random session keys independent of H and passwords (Lines 40 and 79).

If GUESS does not happen in both game, then these changes are conceptual. We have

$$\Pr\left[\mathbf{G}_{11}^{\mathcal{A}} \Rightarrow 1 \mid \neg\mathsf{GUESS} \text{ in } \mathbf{G}_{11}^{\mathcal{A}}\right] = \Pr\left[\mathbf{G}_{12}^{\mathcal{A}} \Rightarrow 1 \mid \neg\mathsf{GUESS} \text{ in } \mathbf{G}_{12}^{\mathcal{A}}\right]$$

We claim that GUESS happens in $\mathbf{G}_{11}$ if and only if it happens in $\mathbf{G}_{12}$. It is straightforward to see that GUESS happens in $\mathbf{G}_{11}$ then it also happens in $\mathbf{G}_{12}$, since in $\mathbf{G}_{12}$ we records all potential passwords in $\mathcal{L}_{\text{pw}}$ that may trigger GUESS in $\mathbf{G}_{11}$. If GUESS happens in $\mathbf{G}_{12}$, then there exists $\mathsf{pw}_{\mathsf{U},\mathsf{S}} \in \mathcal{L}_{\text{pw}}$. Moreover, $\mathsf{pw}_{\mathsf{U},\mathsf{S}}$ is recorded in $\mathcal{L}_{\text{pw}}$ only if (U,S) is uncorrupted. By (cf. Lines 37 to 38 and Lines 76 to 78), $\mathsf{pw}_{\mathsf{U},\mathsf{S}} \in \mathcal{L}_{\text{pw}}$ means that there exists (pk, e_1) (resp., (c, e_2)) such that $e_1 \notin \mathcal{L}_1^{\mathsf{U}}$ (resp., $e_2 \notin \mathcal{L}_2^{\mathsf{S}}$) and $(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, \mathsf{pk}, e_1, \text{enc}) \in \mathcal{L}_1$ (resp., $(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, \mathsf{c}, e_2, \text{enc}) \in \mathcal{L}_2$), and thus either $\mathsf{Guess}_{\text{user}}$ or $\mathsf{Guess}_{\text{ser}}$ will be triggered in $\mathbf{G}_{11}$. Therefore, if GUESS happens in $\mathbf{G}_{12}$, then GUESS also happens in $\mathbf{G}_{11}$. Now we have

$$\left|\Pr\left[\mathbf{G}_{11}^{\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\mathbf{G}_{12}^{\mathcal{A}} \Rightarrow 1\right]\right| \le \Pr\left[\mathsf{GUESS} \text{ in } \mathbf{G}_{11}^{\mathcal{A}}\right] = \Pr\left[\mathsf{GUESS} \text{ in } \mathbf{G}_{12}^{\mathcal{A}}\right]$$

Furthermore, we claim that every query to SendResp or SendTerInit will add at most one password into $\mathcal{L}_{\text{pw}}$. That is, at most one password will be recorded in $\mathcal{L}_{\text{pw}}$ in every execution of Lines 37 to 38 or Lines 76 to 78. To see this, suppose that there are two passwords pw and pw' are recorded during a execution of Lines 37 to 38. By Line 37, we have $(\mathsf{pw}, \mathsf{c}, e_2, \text{enc}) \in \mathcal{L}_2$ and $(\mathsf{pw}', \mathsf{c}', e_2, \text{enc}) \in \mathcal{L}_2$ for some c and c'. This means that e_2 is generated by querying $\mathsf{E}_2(\mathsf{pw}, \mathsf{c})$ and $\mathsf{E}_2(\mathsf{pw}', \mathsf{c}')$, which is impossible since we simulate E_2 in a

Game $\mathbf{G}_{12}$

$\mathsf{par} \leftarrow \mathsf{Setup}$
$\mathcal{C} := \emptyset, \beta \leftarrow \{0,1\}$
$\mathsf{Guess}_{\text{user}} := \textbf{false}, \mathsf{Guess}_{\text{ser}} := \textbf{false}$
$b' \leftarrow \mathcal{A}^{O,\mathsf{H},\mathsf{IC}_1,\mathsf{IC}_2}(\mathsf{par})$
for $(\mathsf{U},\mathsf{S}) \in \mathcal{U} \times \mathcal{S}$
  if $(\mathsf{U},\mathsf{S}) \notin \mathcal{C}$: $\mathsf{pw}_{\mathsf{U},\mathsf{S}} \leftarrow \mathcal{PW}$
  if $\exists \mathsf{S}'$ s.t. $\mathsf{pw}_{\mathsf{U},\mathsf{S}'} \in \mathcal{L}_{\text{pw}}$
    $\mathsf{Guess}_{\text{user}} := \textbf{true}$
  if $\exists \mathsf{U}'$ s.t. $\mathsf{pw}_{\mathsf{U}',\mathsf{S}} \in \mathcal{L}_{\text{pw}}$
    $\mathsf{Guess}_{\text{ser}} := \textbf{true}$
return $\beta == b'$

Oracle $\textsc{Execute}(\mathsf{U}, t_1, \mathsf{S}, t_2)$

if $\pi_{\mathsf{U}}^{t_1} \neq \perp$ **or** $\pi_{\mathsf{S}}^{t_2} \neq \perp$
  return $\perp$
$e_1 \xleftarrow{\$} \mathcal{E}_1 \backslash \mathcal{T}_1, \mathcal{T}_1 := \mathcal{T}_1 \cup \{e_1\}$
$e_2 \xleftarrow{\$} \mathcal{E}_2 \backslash \mathcal{T}_2, \mathcal{T}_2 := \mathcal{T}_2 \cup \{e_2\}$
$\mathrm{ctxt} := (\mathsf{U},\mathsf{S},e_1,e_2), \mathsf{SK} \xleftarrow{\$} \mathcal{SK}$
$\pi_{\mathsf{U}}^{t_1} := ((\perp,\perp,e_1), \mathrm{ctxt}, \mathsf{SK}, \textbf{true})$
$\pi_{\mathsf{S}}^{t_2} := ((\perp,\perp,e_2), \mathrm{ctxt}, \mathsf{SK}, \textbf{true})$
$(\pi_{\mathsf{U}}^{t_1}.\mathtt{fr}, \pi_{\mathsf{S}}^{t_2}.\mathtt{fr}) := (\textbf{true}, \textbf{true})$
return $(\mathsf{U}, e_1, \mathsf{S}, e_2)$

Oracle $\textsc{Corrupt}(\mathsf{U},\mathsf{S})$

if $(\mathsf{U},\mathsf{S}) \in \mathcal{C}$: **return** $\perp$
$\mathcal{C} := \mathcal{C} \cup \{(\mathsf{U},\mathsf{S})\}$
$\mathsf{pw}_{\mathsf{U},\mathsf{S}} \leftarrow \mathcal{PW}$
return $\mathsf{pw}_{\mathsf{U},\mathsf{S}}$

Oracle $\mathsf{E}_1(\mathsf{pw},\mathsf{pk})$

if $\exists(\mathsf{pw},\mathsf{pk},e_1,*) \in \mathcal{L}_1$: **return** e_1
$e_1 \xleftarrow{\$} \mathcal{E}_1 \backslash \mathcal{T}_1, \mathcal{T}_1 := \mathcal{T}_1 \cup \{e_1\}$
$\mathcal{L}_1 := \mathcal{L}_1 \cup (\mathsf{pw},\mathsf{pk},e_1,\mathrm{enc})$
return e_1

Oracle $\mathsf{E}_2(\mathsf{pw},\mathsf{c})$

if $\exists(\mathsf{pw},\mathsf{c},e_2,*) \in \mathcal{L}_2$: **return** e_2
$e_2 \xleftarrow{\$} \mathcal{E}_2 \backslash \mathcal{T}_2, \mathcal{T}_2 := \mathcal{T}_2 \cup \{e_2\}$
$\mathcal{L}_2 := \mathcal{L}_2 \cup (\mathsf{pw},\mathsf{c},e_2,\mathrm{enc})$
return e_2

Oracle $\mathsf{D}_1(\mathsf{pw},e_1)$

if $\exists(\mathsf{pw},\mathsf{pk},e_1,*) \in \mathcal{L}_1$
  return pk
$(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KG}$
$\mathcal{L}_{\mathsf{key}} := \mathcal{L}_{\mathsf{key}} \cup \{(\mathsf{pk},\mathsf{sk})\}$
$\mathcal{L}_1 := \mathcal{L}_1 \cup \{(\mathsf{pw},\mathsf{pk},e_1,\mathrm{dec})\}$
return pk

Oracle $\mathsf{D}_2(\mathsf{pw},e_2)$

if $\exists(\mathsf{pw},\mathsf{c},e_2,*) \in \mathcal{L}_2$: **return** c
$\mathsf{c} \xleftarrow{\$} \mathcal{C}, \mathcal{L}_2 := \mathcal{L}_2 \cup (\mathsf{pw},\mathsf{c},e_2,\mathrm{dec})$
return c

Oracle $\textsc{SendInit}(\mathsf{U}, t_1, \mathsf{S})$

if $\pi_{\mathsf{U}}^{t_1} \neq \perp$: **return** $\perp$
$e_1 \xleftarrow{\$} \mathcal{E}_1 \backslash \mathcal{T}_1, \mathcal{T}_1 := \mathcal{T}_1 \cup \{e_1\}$
$\mathcal{L}_1^{\mathsf{U}} := \mathcal{L}_1^{\mathsf{U}} \cup \{e_1\}$
$\pi_{\mathsf{U}}^{t_1} := ((\perp,\perp,e_1),(\mathsf{U},\mathsf{S},e_1,\perp),\perp,\perp)$
$\pi_{\mathsf{U}}^{t_1}.\mathtt{fr} := \textbf{false}$
return (U, e_1)

Oracle $\textsc{SendResp}(\mathsf{S}, t_2, \mathsf{U}, e_1)$

$\pi_{\mathsf{S}}^{t_2} \neq \perp$: **return** $\perp$
if $(\mathsf{U},\mathsf{S}) \in \mathcal{C}$
  $\pi_{\mathsf{S}}^{t_2}.\mathtt{fr} := \textbf{false}$
  $\mathsf{pk} := \mathsf{D}_1(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, e_1)$
  $(\mathsf{c},\mathsf{k}) \leftarrow \mathsf{Encaps}(\mathsf{pk})$
  $e_2 := \mathsf{E}_2(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, \mathsf{c}), \mathrm{ctxt} := (\mathsf{U},\mathsf{S},e_1,e_2)$
  $\mathsf{SK} := \mathsf{H}(\mathrm{ctxt},\mathsf{pk},\mathsf{c},\mathsf{k},\mathsf{pw}_{\mathsf{U},\mathsf{S}})$
else
  $\pi_{\mathsf{S}}^{t_2}.\mathtt{fr} := \textbf{true}, \mathsf{SK} \xleftarrow{\$} \mathcal{SK}$
  if $e_1 \notin \mathcal{L}_1^{\mathsf{U}}$
    for $(\mathsf{pw},\mathsf{pk})$ s.t. $(\mathsf{pw},\mathsf{pk},e_1,\mathrm{enc}) \in \mathcal{L}_1$
      $\mathcal{L}_{\text{pw}} := \mathcal{L}_{\text{pw}} \cup \{\mathsf{pw}\}$
  $e_2 \xleftarrow{\$} \mathcal{E}_2 \backslash \mathcal{T}_2, \mathcal{T}_2 := \mathcal{T}_2 \cup \{e_2\}$
$\pi_{\mathsf{S}}^{t_2} := ((\perp,\perp,e_2), \mathrm{ctxt}, \mathsf{SK}, \textbf{true})$
$\mathcal{L}_2^{\mathsf{S}} := \mathcal{L}_2^{\mathsf{S}} \cup \{e_2\}$
return (S, e_2)

Oracle $\textsc{SendTerInit}(\mathsf{U}, t_1, \mathsf{S}, e_2)$

if $\pi_{\mathsf{U}}^{t_1} = \perp$ **and** $\pi_{\mathsf{U}}^{t_1}.\mathtt{tr} \neq (\mathsf{U},\mathsf{S},*,*)$
  return $\perp$
if $\exists t_2$ s.t. $\pi_{\mathsf{S}}^{t_2}.\mathtt{fr} = \textbf{true}$
  and $\pi_{\mathsf{S}}^{t_2}.\mathtt{tr} = (\mathsf{U},\mathsf{S},e_1,e_2)$
  $\pi_{\mathsf{U}}^{t_1}.\mathtt{fr} := \textbf{true}, \mathsf{SK} := \pi_{\mathsf{S}}^{t_2}.\mathtt{key}$
else
  $\mathrm{ctxt} := (\mathsf{U},\mathsf{S},e_1,e_2)$
  if $(\mathsf{U},\mathsf{S}) \notin \mathcal{C}$
    $\pi_{\mathsf{U}}^{t_1}.\mathtt{fr} := \textbf{true}, \mathsf{SK} \xleftarrow{\$} \mathcal{SK}$
    if $e_2 \notin \mathcal{L}_2^{\mathsf{S}}$
      for (pw,c) s.t. $(\mathsf{pw},\mathsf{c},e_2,\mathrm{enc}) \in \mathcal{L}_2$
        $\mathcal{L}_{\text{pw}} := \mathcal{L}_{\text{pw}} \cup \{\mathsf{pw}\}$
  else
    $\pi_{\mathsf{U}}^{t_1}.\mathtt{fr} := \textbf{false}$
    $\mathsf{pk} := \mathsf{D}_1(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, e_1)$
    Retrieve sk s.t. $(\mathsf{pk},\mathsf{sk}) \in \mathcal{L}_{\mathsf{key}}$
    $\mathsf{c} := \mathsf{D}_2(\mathsf{pw}_{\mathsf{U},\mathsf{S}}, e_2), \mathsf{k} := \mathsf{Decaps}(\mathsf{sk},\mathsf{c})$
    $\mathsf{SK} := \mathsf{H}(\mathrm{ctxt},\mathsf{pk},\mathsf{c},\mathsf{k},\mathsf{pw}_{\mathsf{U},\mathsf{S}})$
$\pi_{\mathsf{U}}^{t_1}.(\mathtt{tr},\mathtt{key},\mathtt{acc}) := (\mathrm{ctxt}, \mathsf{SK}, \textbf{true})$
return true

Oracle $\mathsf{H}(\mathsf{U},\mathsf{S},e_1,e_2,\mathsf{pk},\mathsf{c},\mathsf{k},\mathsf{pw})$

if $\mathcal{L}_{\mathsf{H}}[\mathsf{U},\mathsf{S},e_1,e_2,\mathsf{pk},\mathsf{c},\mathsf{k},\mathsf{pw}] = \perp$
  $\mathcal{L}_{\mathsf{H}}[\mathsf{U},\mathsf{S},e_1,e_2,\mathsf{pk},\mathsf{c},\mathsf{k},\mathsf{pw}] := \mathsf{SK} \xleftarrow{\$} \mathcal{SK}$
return $\mathcal{L}_{\mathsf{H}}[\mathsf{U},\mathsf{S},e_1,e_2,\mathsf{pk},\mathsf{c},\mathsf{k},\mathsf{pw}]$

Fig. 15. Final game $\mathbf{G}_{12}$ in proving Theorem 1. $\mathcal{A}$ has access to the set of PAKE oracles $\{\textsc{Execute}, \textsc{SendInit}, \textsc{SendResp}, \textsc{SendTerInit}, \textsc{Corrupt}, \textsc{Reveal}, \textsc{Test}\}$, random oracle H, and ideal ciphers $\mathsf{IC}_1 = (\mathsf{E}_1, \mathsf{D}_1)$ and $\mathsf{IC}_2 = (\mathsf{E}_2, \mathsf{D}_2)$. Oracles $\textsc{Reveal}$ and $\textsc{Test}$ are the same as in $\mathbf{G}_1$ (cf. Fig. 8) so we omit their description here.

collision-free way. Similar argument applies for Lines 76 to 78. Therefore, every query to SendResp or SendTerInit will add at most one password into $\mathcal{L}_{\text{pw}}$.

Now we can bound the happening probability of GUESS in $\mathbf{G}_{12}$. A clean description of $\mathbf{G}_{12}$ is given in Fig. 15. In $\mathbf{G}_{12}$, passwords of uncorrupted user-server pairs are undefined before $\mathcal{A}$ issues Corrupt queries or ends with output b'. Moreover, oracles Execute, SendInit, SendResp, and SendTerInit can be simulated without using uncorrupted passwords. Therefore, uncorrupted passwords are perfectly hidden from $\mathcal{A}$'s view. Since $\mathcal{A}$ issues S queries to SendResp and SendTerInit, we have $|\mathcal{L}_{\text{pw}}| \leq S$ and

$$\Pr\left[\mathsf{GUESS} \text{ in } \mathbf{G}_{12}^{\mathcal{A}}\right] \leq \frac{S}{|\mathcal{PW}|}$$

All fresh instances in $\mathbf{G}_{12}$ will accept independently and uniformly random session keys, so we also have

$$\Pr\left[\mathbf{G}_{12}^{\mathcal{A}} \Rightarrow 1\right] = \frac{1}{2}$$

Combining all the probability differences in the games sequence, we have

$$\begin{aligned}
\mathsf{Adv}_{\Pi}^{\mathsf{BPR}}(\mathcal{A}) \leq\ & \frac{S}{|\mathcal{PW}|} + \mathsf{Adv}_{\mathsf{KEM}}^{q_1\text{-}\mathsf{FUZZY}}(\mathcal{B}_1) + \mathsf{Adv}_{\mathsf{KEM}}^{(S,q_2+S)\text{-}\mathsf{OW\text{-}rPCA}}(\mathcal{B}_4) \\
& + \mathsf{Adv}_{\mathsf{KEM}}^{(S,1)\text{-}\mathsf{OW\text{-}PCA}}(\mathcal{B}_2) + \mathsf{Adv}_{\mathsf{KEM}}^{(S+q_2,S)\text{-}\mathsf{OW\text{-}PCA}}(\mathcal{B}_5) \\
& + \mathsf{Adv}_{\mathsf{KEM}}^{(S,1)\text{-}\mathsf{ANO}}(\mathcal{B}_3) + \mathsf{Adv}_{\mathsf{KEM}}^{(S+q_1,S)\text{-}\mathsf{ANO}}(\mathcal{B}_6) + S \cdot \delta \\
& + S^2(\eta_{pk} + \eta_{ct}) + \frac{(q_1^2 + S^2)}{|\mathcal{E}_1|} + \frac{(q_2^2 + S^2)}{|\mathcal{E}_2|} + \frac{q_1^2}{|\mathcal{PK}|} + \frac{q_2^2}{|\mathcal{C}|} + \frac{(q_{\mathsf{H}}^2 + S^2)}{|\mathcal{SK}|}
\end{aligned}$$

5 Instantiations of the Underlying KEM

5.1 Direct Diffie-Hellman-Based Constructions

<u>Diffie-Hellman Assumptions.</u> We recall the multi-user and multi-challenge strong Diffie-Hellman assumption. Let $\mathcal{G}$ be a group generation algorithm that on input security parameters outputs a group description $(\mathbb{G}, g, p)$, where p is an odd prime and $\mathbb{G}$ is a p-order group with generator g.

Definition 15 (Multi-Instance stDH [3]). *Let N and μ be integers. We say the stDH problem is hard on $\mathcal{G}$, if for any $\mathcal{A}$, the (N,μ)-stDH advantage of $\mathcal{A}$ against $\mathcal{G}$*

$$\mathsf{Adv}_{\mathcal{G}}^{(N,\mu)\text{-}\mathsf{stDH}}(\mathcal{A}) := \Pr\left[\mathsf{stDH}_{\mathcal{G}}^{(N,\mu),\mathcal{A}} \Rightarrow 1\right].$$

is negligible, where $\mathsf{stDH}_{\mathcal{G}}^{(N,\mu),\mathcal{A}}$ is defined in Fig. 16.

GAME $\mathsf{stDH}_{\mathcal{G}}^{(N,\mu),\mathcal{A}}$	**Oracle** $\textsc{Pco}(i, Y, Z)$
$\mathsf{par} := (\mathbb{G}, g, p) \leftarrow \mathcal{G}$	**if** $\mathbf{X}[i] = \perp$
for $i \in [N]$	$\quad$ **return** $\perp$
$\quad x_i \xleftarrow{\$} \mathbb{Z}_p, \mathbf{X}[i] := X_i := g^{x_i}$	**return** $Z == Y^{x_i}$
for $j \in [\mu]$:	
$\quad y_j \xleftarrow{\$} \mathbb{Z}_p, \mathbf{Y}[j] := Y_j := g^{y_j}$	
$(i^*, j^*, Z^*) \leftarrow \mathcal{A}^{\textsc{stDH}}(\mathsf{par}, \mathbf{X}, \mathbf{Y})$	
return $Z^* = Y_{j^*}^{x_{i^*}}$	

Fig. 16. Security games OW-PCA and OW-rPCA for KEM scheme KEM.

KG_1	$\mathsf{Encaps}_1(\mathsf{pk})$	$\mathsf{Decaps}_1(\mathsf{sk}, R)$
$x \xleftarrow{\$} \mathbb{Z}_p$	$r \xleftarrow{\$} \mathbb{Z}_p$	**parse** $(x, \mathsf{pk}) =: \mathsf{sk}$
$X := g^x$	$R := g^r \in \mathbb{G}$	**parse** $R =: \mathsf{c}$
$\mathsf{pk} := X$	$\mathsf{k} := \mathsf{H}(\mathsf{pk}, R, X^r)$	$\mathsf{k} := \mathsf{H}(\mathsf{pk}, R, R^x)$
$\mathsf{sk} := (x, \mathsf{pk})$	$\mathsf{c} := R$	**return** k
return $(\mathsf{pk}, \mathsf{sk})$	**return** (c, k)	

Fig. 17. KEM scheme $\mathsf{KEM}_{\mathsf{stDH}} = (\mathsf{Setup}_1, \mathsf{KG}_1, \mathsf{Encaps}_1, \mathsf{Decaps}_1)$.

<u>Construction based on strong DH.</u> In Fig. 17, we construct a KEM scheme $\mathsf{KEM}_{\mathsf{stDH}}$ with plaintext space $\mathbb{G}$ and ciphertext space of $\mathbb{G}$. $\mathsf{KEM}_{\mathsf{stDH}}$ is essentially the hashed ElGamal KEM [3,17].

$\mathsf{KEM}_{\mathsf{stDH}}$ has perfect public key fuzzyness and ciphertext anonymity (even under PCA). This is because $X \xleftarrow{\$} \mathbb{G}$ is equivalent to $(x \xleftarrow{\$} \mathbb{Z}_p, X := g^x)$. Therefore, we have

$$\mathsf{Adv}_{\mathsf{KEM}_{\mathsf{stDH}}}^{(N,\mu)\text{-}\mathsf{ANO}}(\mathcal{A}) = 0,\ \mathsf{Adv}_{\mathsf{KEM}_{\mathsf{stDH}}}^{N\text{-}\mathsf{FUZZY}}(\mathcal{A}) = 0$$

for any integers N and μ, and adversary $\mathcal{A}$ (even unbounded).

It is well-known that the hash ElGamal KEM is tightly IND-CCA secure (which implies OW-PCA security) if the $(1, 1)$-stDH assumption holds [15]. By using the random self-reducibility of Diffie-Hellman assumption, one can show that the (N, μ)-OW-PCA security can be tightly reduced to the $(1, 1)$-stDH assumption.

5.2 Generic Constructions

Let $\mathsf{PKE}_0 = (\mathsf{KG}_0, \mathsf{Enc}_0, \mathsf{Dec}_0)$ be a PKE scheme with public key space $\mathcal{PK}$, message space $\mathcal{M}$, randomness space $\mathcal{R}$, and ciphertext space $\mathcal{C}$. Let ℓ and L be integers. Let $\mathsf{G} : \mathcal{PK} \times \mathcal{M} \to \mathcal{R}$, $\mathsf{H} : \mathcal{PK} \times \mathcal{M} \times \mathcal{C} \to \{0,1\}^L$, and $\mathsf{H}' : \mathcal{PK} \times \{0,1\}^\ell \times \mathcal{C} \to \{0,1\}^L$ be hash functions. Let $\mathsf{PKE}_0 = (\mathsf{Setup}_0, \mathsf{KG}_0, \mathsf{Enc}_0, \mathsf{Dec}_0)$ be a PKE scheme. In Fig. 18, we define a generic transformation for KEM schemes. We denote such transformation as $\mathsf{KEM} = \mathsf{TU}^{\not\perp}[\mathsf{PKE}_0, \mathsf{G}, \mathsf{H}, \mathsf{H}']$. $\mathsf{TU}^{\not\perp}$ is essentially a combination of the T transformation and the $\mathsf{U}^{\not\perp}$ transformation in [21]. KEM has the same public key space and ciphertext space with PKE_0. The Setup algorithm of KEM is the same as the one of PKE_0.

$\underline{\mathsf{KG}(\mathsf{par})}$	$\underline{\mathsf{Encaps}(\mathsf{pk})}$	$\underline{\mathsf{Decaps}((\mathsf{pk},\mathsf{sk},s),\mathsf{c})}$
$(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KG}_0(\mathsf{par})$	$m \xleftarrow{\$} \mathcal{M}'$	$m' := \mathsf{Dec}_0(\mathsf{sk},\mathsf{c})$
$s \xleftarrow{\$} \{0,1\}^\ell$	$r := \mathsf{G}(\mathsf{pk},m)$	**if** $m' \neq \perp$
$\mathsf{sk}' := (\mathsf{pk},\mathsf{sk},s)$	$\mathsf{c} := \mathsf{Enc}_0(\mathsf{pk},m;r)$	**and** $\mathsf{c} = \mathsf{Enc}_0(\mathsf{pk},m';\mathsf{G}(\mathsf{pk},m'))$
return $(\mathsf{pk},\mathsf{sk}')$	$\mathsf{k} := \mathsf{H}(\mathsf{pk},\mathsf{c},m)$	$\quad \mathsf{k} := \mathsf{H}(\mathsf{pk},\mathsf{c},m')$
	return (c,k)	**else** $\mathsf{k} := \mathsf{H}'(\mathsf{pk},\mathsf{c},s)$
		return k

Fig. 18. KEM scheme $\mathsf{KEM} = (\mathsf{Setup},\mathsf{KG},\mathsf{Encaps},\mathsf{Decaps})$ from the generic transformation $\mathsf{TU}^{\not\perp}[\mathsf{PKE}_0,\mathsf{G},\mathsf{H},\mathsf{H}']$, where G,H, and H' are hash functions, $\mathsf{PKE}_0 = (\mathsf{Setup}_0, \mathsf{KG}_0,\mathsf{Enc}_0,\mathsf{Dec}_0)$ is a PKE scheme, and $\mathsf{Setup} = \mathsf{Setup}_0$.

Correctness of KEM. We follow the correctness proof of [21, Theorem 3.1].

Decaps has decapsulation error if its input is $\mathsf{c} = \mathsf{Enc}_0(\mathsf{pk},m';\mathsf{G}(\mathsf{pk},m'))$ for some m' and $\mathsf{Dec}_0(\mathsf{sk},\mathsf{c}) \neq m'$. If PKE_0 is $(1-\delta_{\mathsf{PKE}_0})$-correct, such event happens within probability $q_\mathsf{G} \cdot \delta_{\mathsf{PKE}_0}$ if we treat G as a random oracle and assume G will be queried at most q_G times. Therefore, KEM is $(1-q_\mathsf{G}\cdot\delta_{\mathsf{PKE}_0})$-correct.

Security. In Theorems 2 to 4, we show if PKE_0 has fuzzy public keys and PR-CPA security, then KEM has fuzzy public keys, anonymous ciphertexts (under PCA attacks), and OW-(r)PCA security.

It is easy to see $\mathsf{TU}^{\not\perp}$ transformation preserves the public key fuzzyness of the underlying PKE.

Theorem 2. *Let N be the number of users. If PKE_0 has fuzzy public keys, then $\mathsf{KEM} = \mathsf{TU}^{\not\perp}[\mathsf{PKE}_0,\mathsf{G},\mathsf{H},\mathsf{H}']$ in Fig. 18 also has fuzzy public keys. Concretely, for any adversary $\mathcal{A}$ against KEM, there exists an adversary $\mathcal{B}$ with $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B})$ and*

$$\mathsf{Adv}_{\mathsf{KEM}}^{N\text{-}\mathsf{FUZZY}}(\mathcal{A}) \leq \mathsf{Adv}_{\mathsf{PKE}_0}^{N\text{-}\mathsf{FUZZY}}(\mathcal{B})$$

Theorems 3 and 4 show shat if PKE_0 is PR-CPA secure, then $\mathsf{KEM} = \mathsf{TU}^{\not\perp}[\mathsf{PKE}_0,\mathsf{G},\mathsf{H},\mathsf{H}']$ has OW-CPA security and ciphertext anonymity under PCA attacks. For readability, we postpone their proofs to our full version [28].

Theorem 3. *Let N and μ be the numbers of users and challenge ciphertexts per user. If PKE_0 is PR-CPA secure and $(1-\delta)$-correct and G,H, and H' be random oracles, then $\mathsf{KEM} = \mathsf{TU}^{\not\perp}[\mathsf{PKE}_0,\mathsf{G},\mathsf{H},\mathsf{H}']$ has anonymous ciphertext under PCA attacks (cf. Definition 7).*

Concretely, for any $\mathcal{A}$ against KEM*, there exists $\mathcal{B} = (\mathcal{B}_0, \mathcal{B}_1)$ with $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B})$ and*

$$\mathsf{Adv}_{\mathsf{KEM}}^{(N,\mu)\text{-}\mathsf{ANO}}(\mathcal{A}) \leq 2\mathsf{Adv}_{\mathsf{PKE}_0}^{(N,\mu)\text{-}\mathsf{PR\text{-}CPA}}(\mathcal{B}) + 2Nq_{\mathsf{G}} \cdot \delta + \frac{N\mu q_{\mathsf{G}}}{|\mathcal{M}|} + \frac{2N(q_{\mathsf{H}'} + q_{\mathrm{Pco}})}{2^{\ell}} + \frac{N^2\mu^2 + q_{\mathsf{G}}^2}{|\mathcal{R}|} + \frac{2N^2\mu^2 + q_{\mathsf{H}}^2 + q_{\mathsf{H}'}^2}{2^L},$$

where $q_{\mathsf{G}}, q_{\mathsf{H}}, q_{\mathsf{H}'}$, and q_{Pco} are the numbers of $\mathcal{A}$'s queries to $\mathsf{G}, \mathsf{H}, \mathsf{H}'$, *and* Pco.

Theorem 4. *Let N and μ be the numbers of users and challenge ciphertexts per user. If* PKE_0 *is PR-CPA secure and* G, H, *and* H' *be random oracles, then* $\mathsf{KEM} = \mathsf{TU}^{\not\perp}[\mathsf{PKE}_0, \mathsf{G}, \mathsf{H}, \mathsf{H}']$ *is OW-PCA secure.*

Concretely, for any $\mathcal{A}$ against KEM*'s (N, μ)-*OW-PCA *security, there exists $\mathcal{B}$ with $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B})$ and*

$$\mathsf{Adv}_{\mathsf{KEM}}^{(N,\mu)\text{-}\mathsf{OW\text{-}PCA}}(\mathcal{A}) \leq 2\mathsf{Adv}_{\mathsf{PKE}_0}^{(N,\mu)\text{-}\mathsf{PR\text{-}CPA}}(\mathcal{B}) + 2Nq_{\mathsf{G}} \cdot \delta + \frac{N\mu(q_{\mathsf{G}} + q_{\mathsf{H}})}{|\mathcal{M}|} + \frac{2N(q_{\mathsf{H}'} + q_{\mathrm{Pco}})}{2^{\ell}} + \frac{N^2\mu^2 + q_{\mathsf{G}}^2}{|\mathcal{R}|} + \frac{2N^2\mu^2 + q_{\mathsf{H}}^2 + q_{\mathsf{H}'}^2}{2^L},$$

where $q_{\mathsf{G}}, q_{\mathsf{H}}, q_{\mathsf{H}'}$, and q_{Pco} are the numbers of $\mathcal{A}$'s queries to $\mathsf{G}, \mathsf{H}, \mathsf{H}'$, *and* Pco.

By combining Lemma 1 and Theorems 3 and 4, we have Theorem 5.

Theorem 5. *Let N and μ be the numbers of users and challenge ciphertexts per user. If* PKE_0 *is PR-CPA secure and* G, H, *and* H' *be random oracles, then* $\mathsf{KEM} = \mathsf{TU}^{\not\perp}[\mathsf{PKE}_0, \mathsf{G}, \mathsf{H}, \mathsf{H}']$ *is OW-rPCA secure.*

Concretely, for any $\mathcal{A}$ against KEM*'s (N, μ)-*OW-rPCA *security, there exists $\mathcal{B}$ with $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B})$ and*

$$\mathsf{Adv}_{\mathsf{KEM}}^{(N,\mu)\text{-}\mathsf{OW\text{-}rPCA}}(\mathcal{A}) \leq 4\mathsf{Adv}_{\mathsf{PKE}_0}^{(N,\mu)\text{-}\mathsf{PR\text{-}CPA}}(\mathcal{B}) + 4Nq_{\mathsf{G}} \cdot \delta + \frac{N\mu(2q_{\mathsf{G}} + q_{\mathsf{H}})}{|\mathcal{M}|} + \frac{4N(q_{\mathsf{H}'} + q_{\mathrm{Pco}})}{2^{\ell}} + \frac{2(N^2\mu^2 + q_{\mathsf{G}}^2)}{|\mathcal{R}|} + \frac{2(2N^2\mu^2 + q_{\mathsf{H}}^2 + q_{\mathsf{H}'}^2)}{2^L},$$

where $q_{\mathsf{G}}, q_{\mathsf{H}}, q_{\mathsf{H}'}$, and q_{Pco} are the numbers of $\mathcal{A}$'s queries to $\mathsf{G}, \mathsf{H}, \mathsf{H}'$, *and* Pco.

5.3 Lattice-Based Instantiations

We discuss two lattice-based instantiations of the PAKE protocol Π (Fig. 7). The first one is the well-known Regev's encryption [29,30] which is based on learning with error (LWE) assumption. The second one is the Kyber.PKE scheme [32], which is based on the module LWE (MLWE) assumption. For simplicity, we only discuss the security loss of these schemes (from their assumptions) and the final security loss of Π instantiated with these schemes. For more background about lattices, please refer to [18,29,30,32].

Let λ the security parameter. Let S and q_{IC} be the number of session and the number of $\mathcal{A}$'s queries to ideal ciphers $(\mathsf{IC}_1, \mathsf{IC}_2)$ in Fig. 7. Let ϵ_{LWE} and ϵ_{mlwe} be the best computational advantage against the LWE and MLWE assumptions, respectively. We use $\mathsf{negl}(\lambda)$ to denote negligible (about λ) statistical terms. Such terms do not influence tightness.

<u>Regev Encryption.</u> We use the multi-bit version of Regev's encryption, denoted as $\mathsf{PKE}_{\mathsf{Regev}}$, in [29]. As shown in [29, Lemma 7.3, Lemma 7.4], the public keys of this scheme are indistinguishable from random by using a LWE problem instance, and the ciphertexts are pseudorandom under random public keys. Suppose this scheme encrypts $\Theta(\lambda)$ bits, then we have

$$\mathsf{Adv}^{N\text{-}\mathsf{FUZZY}}_{\mathsf{PKE}_{\mathsf{Regev}}}(\mathcal{A}) \leq O(N\lambda) \cdot \epsilon_{\mathsf{LWE}}, \ \mathsf{Adv}^{(N,\mu)\text{-}\mathsf{PR}\text{-}\mathsf{CPA}}_{\mathsf{PKE}_{\mathsf{Regev}}}(\mathcal{A}) \leq O(N\lambda) \cdot \epsilon_{\mathsf{LWE}} + \mathsf{negl}(\lambda)$$

We can use the $\mathsf{TU}^{\not\perp}$ transformation to transform $\mathsf{PKE}_{\mathsf{Regev}}$ into a KEM scheme and then use the KEM scheme to instantiate Π (Fig. 7). By plugging these bounds into Theorems 3 to 5 and then Theorem 1, we have

$$\mathsf{Adv}^{\mathsf{BPR}}_{\Pi[\mathsf{PKE}_{\mathsf{Regev}}]}(\mathcal{A}) \leq O(\lambda \cdot (q_{\mathsf{IC}} + S)) \cdot \epsilon_{\mathsf{LWE}}$$

<u>Kyber PKE.</u> We consider the Kyber.CPAPKE scheme (denoted as $\mathsf{PKE}_{\mathsf{kyber}}$) in [32]. The pseudorandomness and fuzzyness proofs of $\mathsf{PKE}_{\mathsf{kyber}}$ are the same as in [8, Lemmata 1 and 2, Corollary 1]. Since the MLWE assumption does not have random self-reducibility, we can use a standard hybrid argument to extend such proofs to multi-user-challenge setting. We have

$$\mathsf{Adv}^{N\text{-}\mathsf{FUZZY}}_{\mathsf{PKE}_{\mathsf{Regev}}}(\mathcal{A}) \leq N \cdot \epsilon_{\mathsf{mlwe}}, \ \mathsf{Adv}^{(N,\mu)\text{-}\mathsf{PR}\text{-}\mathsf{CPA}}_{\mathsf{PKE}_{\mathsf{Regev}}}(\mathcal{A}) \leq N\mu \cdot 2\epsilon_{\mathsf{mlwe}}$$

By using the $\mathsf{TU}^{\not\perp}$ transformation, we can transform $\mathsf{PKE}_{\mathsf{kyber}}$ into a KEM scheme. Then we use the KEM scheme to instantiate Π (Fig. 7). By Theorems 1 and 3 to 5, we have

$$\mathsf{Adv}^{\mathsf{BPR}}_{\Pi[\mathsf{PKE}_{\mathsf{kyber}}}(\mathcal{A}) \leq O(S \cdot (q_{\mathsf{IC}} + S)) \cdot \epsilon_{\mathsf{mlwe}}$$

References

1. Abdalla, M., Barbosa, M.: Perfect forward security of SPAKE2. Cryptology ePrint Archive, Report 2019/1194 (2019). https://eprint.iacr.org/2019/1194
2. Abdalla, M., Barbosa, M., Bradley, T., Jarecki, S., Katz, J., Xu, J.: Universally composable relaxed password authenticated key exchange. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part I. LNCS, vol. 12170, pp. 278–307. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56784-2_10
3. Abdalla, M., Bellare, M., Rogaway, P.: The oracle Diffie-Hellman assumptions and an analysis of DHIES. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 143–158. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45353-9_12

4. Abdalla, M., Eisenhofer, T., Kiltz, E., Kunzweiler, S., Riepel, D.: Password-authenticated key exchange from group actions. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part II. LNCS, vol. 13508, pp. 699–728. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-15979-4_24
5. Abdalla, M., Fouque, P.-A., Pointcheval, D.: Password-based authenticated key exchange in the three-party setting. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 65–84. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30580-4_6
6. Abdalla, M., Pointcheval, D.: Simple password-based encrypted key exchange protocols. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 191–208. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30574-3_14
7. Becerra, J., Iovino, V., Ostrev, D., Šala, P., Škrobot, M.: Tightly-secure PAK(E). In: Capkun, S., Chow, S.S.M. (eds.) CANS 2017. LNCS, vol. 11261, pp. 27–48. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-02641-7_2
8. Beguinet, H., Chevalier, C., Pointcheval, D., Ricosset, T., Rossi, M.: GeT a CAKE: generic transformations from key encaspulation mechanisms to password authenticated key exchanges. ACNS 2023 (2023). https://eprint.iacr.org/2023/470
9. Bellare, M., Palacio, A.: GQ and Schnorr identification schemes: proofs of security against impersonation under active and concurrent attacks. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 162–177. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45708-9_11
10. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45539-6_11
11. Bellare, M., Rogaway, P.: The security of triple encryption and a framework for code-based game-playing proofs. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 409–426. Springer, Heidelberg (2006). https://doi.org/10.1007/11761679_25
12. Bellovin, S.M., Merritt, M.: Encrypted key exchange: password-based protocols secure against dictionary attacks. In: 1992 IEEE Symposium on Security and Privacy, pp. 72–84. IEEE Computer Society Press (1992)
13. Benhamouda, F., Blazy, O., Ducas, L., Quach, W.: Hash proof systems over lattices revisited. In: Abdalla, M., Dahab, R. (eds.) PKC 2018, Part II. LNCS, vol. 10770, pp. 644–674. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-76581-5_22
14. Bernstein, D.J., Persichetti, E.: Towards KEM unification. Cryptology ePrint Archive, Report 2018/526 (2018). https://eprint.iacr.org/2018/526
15. Bhattacharyya, R.: Memory-tight reductions for practical key encapsulation mechanisms. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020, Part I. LNCS, vol. 12110, pp. 249–278. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45374-9_9
16. Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P.: Universally composable password-based key exchange. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 404–421. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_24
17. Cash, D., Kiltz, E., Shoup, V.: The twin Diffie-Hellman problem and applications. In: Smart, N. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 127–145. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78967-3_8

18. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Ladner, R.E., Dwork, C. (eds.) 40th ACM STOC, pp. 197–206. ACM Press (2008)
19. Haase, B., Labrique, B.: AuCPace: efficient verifier-based PAKE protocol tailored for the IIoT. IACR TCHES **2019**(2), 1–48 (2019). https://tches.iacr.org/index.php/TCHES/article/view/7384
20. Hao, F., Ryan, P.: J-PAKE: authenticated key exchange without PKI. Cryptology ePrint Archive, Report 2010/190 (2010). https://eprint.iacr.org/2010/190
21. Hofheinz, D., Hövelmanns, K., Kiltz, E.: A modular analysis of the Fujisaki-Okamoto transformation. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017, Part I. LNCS, vol. 10677, pp. 341–371. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70500-2_12
22. Jablon, D.P.: Strong password-only authenticated key exchange. SIGCOMM Comput. Commun. Rev. **26**(5), 5–26 (1996). https://doi.org/10.1145/242896.242897
23. Katz, J., Vaikuntanathan, V.: Smooth projective hashing and password-based authenticated key exchange from lattices. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 636–652. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10366-7_37
24. Liu, X., Liu, S., Han, S., Gu, D.: EKE meets tight security in the Universally Composable framework. In: Boldyreva, A., Kolesnikov, V. (eds.) PKC 2023, Part I. LNCS, vol. 13940, pp. 685–713. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-31368-4_24
25. MacKenzie, P.: The PAK suite: protocols for password-authenticated key exchange (2002)
26. Okamoto, T., Pointcheval, D.: The gap-problems: a new class of problems for the security of cryptographic schemes. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 104–118. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44586-2_8
27. Okamoto, T., Pointcheval, D.: REACT: rapid enhanced-security asymmetric cryptosystem transform. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 159–174. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45353-9_13
28. Pan, J., Zeng, R.: A generic construction of tightly secure password-based authenticated key exchange. Cryptology ePrint Archive (2023). https://ia.cr/2023/1334
29. Peikert, C., Vaikuntanathan, V., Waters, B.: A framework for efficient and composable oblivious transfer. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 554–571. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85174-5_31
30. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) 37th ACM STOC, pp. 84–93. ACM Press (2005)
31. Santos, B.F.D., Gu, Y., Jarecki, S.: Randomized half-ideal cipher on groups with applications to UC (a)PAKE. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part V. LNCS, vol. 14008, pp. 128–156. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-30589-4_5
32. Schwabe, P., et al.: CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology (2020). https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions
33. Zhang, J., Yu, Yu.: Two-round PAKE from approximate SPH and instantiations from lattices. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part III. LNCS, vol. 10626, pp. 37–67. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70700-6_2