


A Generic Construction of Tightly Secure Password-based Authenticated Key Exchange*

Jiaxin Pan 

Runzhi Zeng 

September 7, 2023

Department of Mathematical Sciences,
NTNU Norwegian University of Science and Technology, Trondheim, Norway
`jiaxin.pan@ntnu.no`, `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

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

Contents

1	Introduction	3
1.1	Our Contribution	4
2	Preliminaries	5
2.1	Key Encapsulation Mechanism	5
2.2	Public-Key Encryption	7
3	Password-based Authenticated Key Exchange	8
3.1	Definition of PAKE	8
3.2	Security Model of PAKE	9
4	Our Generic Construction of PAKE	11
4.1	Proof of Theorem 4.1	11
5	Instantiations of the Underlying KEM	21
5.1	Direct Diffie-Hellman-based Constructions	21
5.2	Generic Constructions	22
5.3	Lattice-based Instantiations	24
A	Omitted Proofs in Section 4.1	28
B	Omitted Proofs in Section 5.2	31
C	On the Universal Composability of Π	34

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 to 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 drawn a lot of attention in the past few years.

After the introduction of Encrypted-Key-Exchange (EKE) protocol by Bellare and Merritt [BM92], many PAKE protocols have been proposed based on variants of the Diffie-Hellman assumptions, including the well-known SPEKE [Jab96], SPEKE2 [AP05], J-PAKE [HR10], and CPace [HL19]. There are only a few exceptions where PAKE is constructed based on *post-quantum assumptions*, such as lattices [KV09, BBDQ18, ZY17] and group actions [AEK⁺22].

SECURITY OF PAKE. 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 [BPR00] or the universal composability (UC) framework [CHK⁺05].

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

$$\varepsilon_{\text{PAKE}} \leq S/|\mathcal{PW}| + L \cdot \varepsilon_{\text{Problem}}, \quad (1)$$

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 Equation (1) for simplicity. Essentially, $S/|\mathcal{PW}|$ is the success probability of online dictionary attacks and Equation (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.

TIGHT SECURITY. We say a security proof for PAKE is tight if L is a small constant. All the aforementioned PAKE protocols are non-tight. For instance, according to the analysis of [BCP⁺23], 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 Com-GA-PAKE $_\ell$ in [AEK⁺22] is even worse, and it contains a square root of the advantage of the underlying assumption (cf. [AEK⁺22, Theorem 2]), due to the Reset Lemma [BP02]. This means even if we set up the underlying assumption with 128-bit security, Com-GA-PAKE $_\ell$ in [AEK⁺22] has only less¹ than 64-bit.

We note that X-GA-PAKE $_\ell$ in [AEK⁺22, 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 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. [BIO⁺17] proved tight security of the three-move PAK protocol [Mac02] using the Gap DH (GDH) assumption [OP01a] in the IND-based model, where the GDH assumption states that the Computational

¹This is because of the additional multiplicative loss factor depending on S and the length of a password in [AEK⁺22, Theorem 2].

DH (CDH) assumption is hard even if the Decisional DH (DDH) assumption is easy. Lately, Abdalla et al. [ABB⁺20] proved tight security of two-move SPAKE2 in the relaxed UC framework under the GDH assumption. Very recently, Liu et al. [LLHG23] carefully used the twinning technique [CKS08] 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 [Reg05] are equivalent.

Regarding the assumption based on group actions, as we discussed earlier, the Com-GA-PAKE_ℓ protocol in [AEK⁺22] needs to rewind an adversary to argue PFS, and by using the Reset Lemma it leads to a very loose bound. Apart from that, Com-GA-PAKE_ℓ 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. [LLHG23] did not provide a formal proof on the PFS of their protocol, but rather an informal remark. In [AEK⁺22], 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 [OP01b].
- 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 security²), 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 [Reg05]) 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. [BCP⁺23]. More details will be provided in “COMPARISON USING KYBER”.
- or *directly* from the strong DH (stDH) assumption in a prime-order group [ABR01]. 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. [LLHG23] in terms of protocol transcripts. In fact, using the twinning technique of Cash et al. [CKS08], 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 [AEK⁺22] and lattices as in [BBDQ18], our construction is compact and does not use “bit-by-bit” approaches. Figure 1 briefly summarizes our approaches.

Our proofs are in the IND-based model (aka, the so-called Bellare-Pointcheval-Rogaway (BPR) model [BPR00]) 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 Supp. Mat. C.

COMPARISON USING KYBER [SAB⁺20]. There are only a few efficient PAKE protocols from lattices. We focus our comparison on the very efficient one by implementing the CAKE in [BCP⁺23] with KYBER.

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

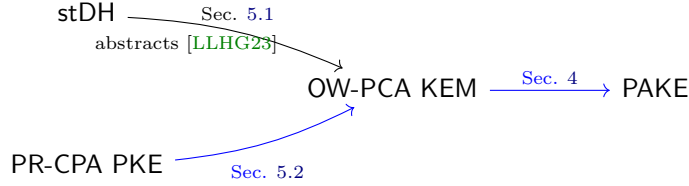


Figure 1: Overview of our construction. All implications are tight, and the blue ones are done via generic constructions. OW-PCA security is the core for our “KEM-to-PAKE” transformation. Please find additional requirements on the KEM in the text.

The reason of not using OCAKE in [BCP⁺23] is because 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 attacks³, 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 [SGJ23], and we leave the formal proof for it as an open problem.

2 Preliminaries

For an integer n , we define the notation $[n] := \{1, \dots, n\}$. 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 $\mathcal{A}^{(\mathcal{O})}$ 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 [BR06] 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 Ev be an (classical) event. We write $\Pr[\text{Ev} : \mathbf{G}]$ to denote the probability that Ev occurs during the game \mathbf{G} . In our security notions throughout the paper, we let N, μ be numbers 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 2.1 (Key Encapsulation Mechanism). A KEM consists of four algorithms (Setup, KG, Encaps, Decaps) and a ciphertext space \mathcal{C} , a randomness space \mathcal{R} , and a KEM key space \mathcal{K} . On input

³More precisely, the argument in [BCP⁺23, page 41] under “Analysis” may not hold true for reply attacks.

GAME $\text{OW-PCA}_{\text{KEM}}^{(N,\mu),\mathcal{A}}$	GAME $\text{OW-rPCA}_{\text{KEM}}^{(N,\mu),\mathcal{A}}$
01 par $\leftarrow \text{Setup}$	10 par $\leftarrow \text{Setup}$
02 for $i \in [N]$	11 for $i \in [N]$
03 $(\text{pk}, \text{sk}) \leftarrow \text{KG}(\text{par})$	12 $(\text{pk}[i], \text{sk}[i]) := (\text{pk}, \text{sk}) \leftarrow \text{KG}(\text{par})$
04 $(\text{pk}[i], \text{sk}[i]) := (\text{pk}, \text{sk})$	13 for $j \in [N \cdot \mu]$:
05 for $j \in [\mu]$:	14 $\text{c}[j] := \text{c} \stackrel{\$}{\leftarrow} \mathcal{C}$
06 $(\text{c}, \text{k}) \leftarrow \text{Encaps}(\text{pk}[i])$	15 $(i, j, \text{k}^*) \leftarrow \mathcal{A}^{\text{PCo}}(\text{pk}, \text{c})$
07 $(\text{c}[i, j], \text{k}[i, j]) := (\text{c}, \text{k})$	16 return $\text{k}^* == \text{Decaps}(\text{sk}[i], \text{c}[j])$
08 $(i, j, \text{k}^*) \leftarrow \mathcal{A}^{\text{PCo}}(\text{pk}, \text{c})$	Oracle $\text{PCo}(i, \text{c}, \text{k})$
09 return $\text{k}^* == \text{Decaps}(\text{sk}[i], \text{c}[i, j])$	17 if $\text{pk}[i] = \perp$
	18 return \perp
	19 return $\text{k} == \text{Decaps}(\text{sk}[i], \text{c})$

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

security parameters, Setup outputs a system parameter par . $\text{KG}(\text{par})$ outputs a public and secret key pair (pk, sk) . The encapsulation algorithm Encaps , on input pk , outputs a ciphertext $\text{c} \in \mathcal{C}$. We also write $\text{c} := \text{Encaps}(\text{pk}; r)$ to indicate the randomness $r \in \mathcal{R}$ explicitly. The decapsulation algorithm Decaps , on input sk and a ciphertext c , outputs a KEM key $\text{k} \in \mathcal{K}$ 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 2.2 (KEM Correctness). Let $\text{KEM} := (\text{Setup}, \text{KG}, \text{Encaps}, \text{Decaps})$ be a KEM scheme and \mathcal{A} be an adversary against KEM. We say KEM is $(1 - \delta)$ -correct if

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

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

Definition 2.3 (Implicit Rejection [BP18]). A KEM scheme $\text{KEM} = (\text{Setup}, \text{KG}, \text{Encaps}, \text{Decaps})$ has implicit rejection if $\text{Decaps}(\text{sk}, \cdot)$ behaves as a pseudorandom function when the input ciphertext is invalid, where $\text{par} \leftarrow \text{Setup}, (\text{pk}, \text{sk}) \leftarrow \text{KG}$, and sk is the key of the pseudorandom function. That is, if an input ciphertext c is invalid, then $\text{Decaps}(\text{sk}, \text{c})$ will output a pseudorandom key k instead of a rejection symbol \perp . A concrete example is shown in Figure 17.

OW-PCA SECURITY. Let $\text{KEM} = (\text{Setup}, \text{KG}, \text{Encaps}, \text{Decaps})$ be a KEM scheme with ciphertext space \mathcal{C} . In Definitions 2.4 and 2.5, we define two variants of one-wayness under plaintext-checking attacks (OW-PCA) security for KEM [OP01b] 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 2.4 (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

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

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

Definition 2.5 (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}

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

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

GAME ANO-PCA $_{\text{KEM},b}^{(N,\mu),\mathcal{A}}$	GAME FUZZY $_{\text{KEM},b}^{N,\mathcal{A}}$
01 par \leftarrow Setup	10 par \leftarrow Setup
02 for $i \in [N]$	11 for $i \in [N]$
03 $(\mathbf{pk}[i], \mathbf{sk}[i]) := (\mathbf{pk}, \mathbf{sk}) \leftarrow \text{KG}(\text{par})$	12 $(\mathbf{pk}_0[i], \mathbf{sk}[i]) := (\mathbf{pk}, \mathbf{sk}) \leftarrow \text{KG}(\text{par})$
04 for $j \in [\mu]$:	13 $\mathbf{pk}_1[i] := \mathbf{pk} \xleftarrow{\$} \mathcal{PK}$
05 $(\mathbf{c}, \mathbf{k}) \leftarrow \text{Encaps}(\mathbf{pk}[i])$	14 $b' \leftarrow \mathcal{A}(\text{par}, \mathbf{pk}_b)$
06 $(\mathbf{c}_0[i, j], \mathbf{k}[i, j]) := (\mathbf{c}, \mathbf{k})$	15 return b'
07 $\mathbf{c}_1[i, j] \xleftarrow{\$} \mathcal{C}$	
08 $b' \leftarrow \mathcal{A}^{\text{PCo}}(\text{par}, \mathbf{pk}, \mathbf{c}_b)$	
09 return b'	

Figure 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 Figure 2.

Definition 2.6 (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

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

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

Definition 2.7 (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

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

where the game $\text{ANO-PCA}_{\text{KEM},b}^{(N,\mu),\mathcal{A}} (b \in \{0,1\})$ is defined in Figure 3. We say KEM has anonymous ciphertexts under PCA attacks (or simply, anonymous ciphertexts) if $\text{Adv}_{\text{KEM}}^{(N,\mu)\text{-ANO}}(\mathcal{A})$ 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 2.8

Lemma 2.8 (OW-PCA + ANO-PCA \Rightarrow 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

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

2.2 Public-Key Encryption

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

Definition 2.9 (PKE Correctness). Let $\text{PKE} := (\text{Setup}, \text{KG}, \text{Enc}, \text{Dec})$ 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

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

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

GAME COR_{PKE}^A	
01	par ← Setup
02	(pk, sk) ← KG(par)
03	m ← \mathcal{A}^O (par, pk, sk)
04	c ← Enc(pk, m)
05	if Dec(sk, c) ≠ m : return 1
06	return 0

Figure 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.

We define fuzzyness for PKE, which is essentially the same as the one for KEM (cf. Definition 2.6).

Definition 2.10 (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

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

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

PSEUDORANDOM CIPHERTEXT. Let $\text{PKE} := (\text{KG}, \text{Enc}, \text{Dec})$ 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 Figure 5.

Definition 2.11 (Multi-user-challenge 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 $\text{PR-CPA}_{\text{PKE},b}^{(N,\mu),\mathcal{A}}(b \in \{0,1\})$ defined in Figure 5. We define the (N, μ) -PR-CPA advantage function

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

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

GAME PR-CPA_{PKE,b}^{(N,μ),A}	
01	par ← Setup
02	for $i \in N$
03	(pk _i , sk _i) ← KG(par), pk[i] := pk _i
04	(m, st) ← \mathcal{A}_0 (par, pk) // m has $N \times \mu$ messages
05	for $i \in [N]$:
06	for $j \in [\mu]$
07	c ₀ [i, j] ← Enc(pk[i], m[i, j]), c ₁ [i, j] $\xleftarrow{\$}$ \mathcal{C}
08	b' ← \mathcal{A}_1 (st, c _b)
09	return b'

Figure 5: Security game PR-CPA for PKE scheme PKE.

3 Password-based Authenticated Key Exchange

3.1 Definition of PAKE

A two-message PAKE protocol $\text{PAKE} := (\text{Setup}, \text{Init}, \text{Resp}, \text{TerInit})$ 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 $\text{par} \leftarrow \text{Setup}$.

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 , which takes the identities U, S and password pw as inputs and outputs a client message M_U and session state st , and then U sends M_U to S . On receiving M_U , S runs the servers derivation algorithm Resp , which takes identities U and S and the received message M_U as input, together with the password pw , to generate a server message M_S and a session key SK_S . S sends M_S to U . Finally, on receiving M_S , U runs the clients derivation algorithm TerInit which inputs U, S , the session state st generated before, the received message M_S , and password pw , to generate a session key 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.

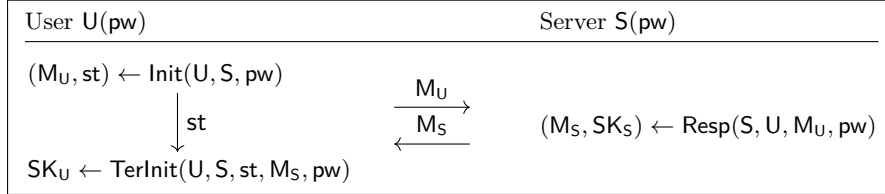


Figure 6: Illustration for a two-message PAKE protocol execution between a user U and a server S .

We define the correctness of PAKE protocols, stating that an honestly execution between user U and server S (with the same password $\text{pw}_{U,S}$) as in Figure 6 will produce the same session key $\text{SK}_U = \text{SK}_S$.

Definition 3.1 (PAKE Correctness). Let $\text{PAKE} := (\text{Setup}, \text{Init}, \text{Resp}, \text{TerInit})$ 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 $\text{par} \leftarrow \text{Setup}$, the following probability is at least ρ .

$$\Pr \left[\text{SK}_U = \text{SK}_S \mid \begin{array}{l} (M_U, \text{st}) \leftarrow \text{Init}(U, S, \text{pw}) \\ (M_S, \text{SK}_S) \leftarrow \text{Resp}(S, U, M_U, \text{pw}) \\ \text{SK}_U \leftarrow \text{TerInit}(U, S, \text{st}, M_S, \text{pw}) \end{array} \right]$$

3.2 Security Model of PAKE

We consider indistinguishability(IND)-based security of PAKE protocols. In this section, we define the multi-test variant of the Bellare-Pointcheval-Rogaway model [BPR00, AFP05, AB19]. 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 with 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 $\mathcal{PW} \subseteq \{0, 1\}^\ell$. Each pair of user and server $U \times S \in \mathcal{U} \times \mathcal{S}$ holds a shared password $\text{pw}_{U,S} \in \mathcal{PW}$.

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 $(e, \text{tr}, \text{key}, \text{acc})$ where

- e is the ephemeral secret chosen by P .
- 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.
- key is the accepted session key of π_P^i .
- 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, $\text{acc} = \perp$ (which means undefined).
- 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 $\pi_P^i.(e, \text{tr}, \text{key}, \text{acc})$. We define partnership via matching instance trace.

Definition 3.2 (Partnering). A user instance $\pi_U^{t_0}$ and a server instance $\pi_S^{t_1}$ are partnered if and only if

$$\pi_U^{t_0}.\text{acc} = \mathbf{true} = \pi_S^{t_1}.\text{acc} \quad \mathbf{and} \quad \pi_U^{t_0}.\text{tr} = \pi_S^{t_1}.\text{tr}$$

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

SECURITY GAME. 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:

- $\text{EXECUTE}(U, t_1, S, t_2)$: This oracle outputs the protocol messages of an honest protocol execution between instances $\pi_U^{t_1}$ and $\pi_S^{t_2}$. By querying this oracle, the adversary launches passive attacks.
- $\text{SENDINIT}, \text{SENDRESP}, \text{SENDTERINIT}$: 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).
- $\text{REVEAL}(P, t)$: By this oracle, the adversary reveals the session key of π_P^t .
- $\text{TEST}(P, t)$: 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 $\pi_P^t.\text{test}$ will be set as **true**.

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

Definition 3.3 (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 instance⁴.
 - (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 3.4 (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

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

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} , $\text{Adv}_{\text{PAKE}}^{\text{BPR}}(\mathcal{A})$ 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 BPR_{PAKE}).

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

4 Our Generic Construction of PAKE

CONSTRUCTION. Let $\text{KEM} = (\text{Setup}, \text{KG}, \text{Encaps}, \text{Decaps})$ 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 $\text{IC}_1 = (\text{E}_1, \text{D}_1)$ be a symmetric encryption with key space \mathcal{PW} , plaintext space \mathcal{PK} , and ciphertext space \mathcal{E}_1 . Let $\text{IC}_2 = (\text{E}_2, \text{D}_2)$ 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 $\Pi = (\text{Init}, \text{Resp}, \text{TerlNit})$ as shown in Figure 6, where \mathcal{SK} is the session key space of PAKE and $\text{H}: \{0, 1\}^* \rightarrow \mathcal{SK}$ is a hash function which is used to derive the session key. The system parameter par is generated by $\text{par} \leftarrow \text{Setup}$.

Alg Init(U, S, pw)	Alg TerlNit($U, S, \text{st}, e_2, \text{pw}$)	Alg Resp(S, U, e_1, pw)
01 $(\text{pk}, \text{sk}) \leftarrow \text{KG}(\text{par})$	05 let $(\text{pk}, \text{sk}, e_1) := \text{st}$	11 $\text{pk} := \text{D}_1(\text{pw}, e_1)$
02 $e_1 := \text{E}_1(\text{pw}, \text{pk})$	06 $c := \text{D}_2(\text{pw}, e_2)$	12 $(c, k) \leftarrow \text{Encaps}(\text{pk})$
03 $\text{st} := (\text{pk}, \text{sk}, e_1)$	07 $k := \text{Decaps}(\text{sk}, c)$	13 $e_2 := \text{E}_2(\text{pw}, c)$
04 return (e_1, st)	08 $\text{ctxt} := (U, S, e_1, e_2)$	14 $\text{ctxt} := (U, S, e_1, e_2)$
	09 $\text{SK} := \text{H}(\text{ctxt}, \text{pk}, c, k, \text{pw})$	15 $\text{SK} := \text{H}(\text{ctxt}, \text{pk}, c, k, \text{pw})$
	10 return SK	16 return (e_2, SK)

Figure 7: Our PAKE protocol Π .

The correctness of Π is dependent on KEM. In Figure 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 4.1 *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 OW-rPCA security (cf. Definitions 2.4 to 2.7), then the PAKE protocol Π in Figure 7 is secure (wrt Definition 3.4).*

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

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

where q_1, q_2, q_H are the numbers of \mathcal{A} queries to IC_1, 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 4.2 (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 [SAB⁺20] in [BCP⁺23, Section 5.2]. More implementation for group-based schemes and lattice-based schemes can be found in [SGJ23].

Remark 4.3 We require KEM to have implicit rejection (cf. Definition 2.3) 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. [HHK17, Definition 2.1]) security to detect whether the c is valid in the proof.

4.1 Proof of Theorem 4.1

Let \mathcal{A} be an adversary against PAKE in the BPR game, where N is the number of parties. Every user-server pair $(U, S) \in \mathcal{U} \times \mathcal{S}$ is associated with a password $\text{pw}_{U, S}$. The game sequences \mathbf{G}_0 - \mathbf{G}_{12} of the proof are given in Figures 8, 9, 11 and 14. The full description of the final game \mathbf{G}_{12} is given in Figure 21.

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., $\text{IC}_1 = (\text{E}_1, \text{D}_1)$, $\text{IC}_2 = (\text{E}_2, \text{D}_2)$, and H . If such a collision happens at any

<p>Game G_0-G_1</p> <p>01 $\text{par} \leftarrow \text{Setup}$</p> <p>02 for $(U, S) \in \mathcal{U} \times \mathcal{S}$</p> <p>03 $\text{pw}_{U,S} \leftarrow \mathcal{PW}$</p> <p>04 $\mathcal{C} := \emptyset$</p> <p>05 $\beta \leftarrow \{0, 1\}$</p> <p>06 $b' \leftarrow \mathcal{A}^{O, H, \text{IC}_1, \text{IC}_2}(\text{par})$</p> <p>07 return $\beta == b'$</p> <p>Oracle REVEAL(P, t)</p> <p>08 if $\pi_P^t.\text{acc} \neq \text{true}$ or $\pi_P^t.\text{test} = \text{true}$</p> <p>09 return \perp</p> <p>10 if $\exists P' \in \mathcal{U} \cup \mathcal{S}, t'$ s.t.</p> <p>11 $\text{Partner}(\pi_P^t, \pi_{P'}^{t'}) = \text{true}$</p> <p>12 and $\pi_{P'}^{t'}.\text{test} = \text{true}$</p> <p>13 return \perp</p> <p>14 for $\forall (P', t')$ s.t. $\pi_{P'}^{t'}.\text{tr} = \pi_P^t.\text{tr}$ // G_1</p> <p>15 $\pi_{P'}^{t'}.\text{fr} := \text{false}$ // G_1</p> <p>16 return $\pi_P^t.\text{key}$</p> <p>Oracle TEST(P, t)</p> <p>17 if $\text{Freshness}(\pi_P^t) = \text{false}$ // G_0</p> <p>18 if $\pi_P^t.\text{fr} = \text{false}$ // G_1</p> <p>19 return \perp</p> <p>20 $\text{SK}_0^* := \text{REVEAL}(P, t), \text{SK}_1^* \xleftarrow{\\$} \mathcal{SK}$</p> <p>21 if $\text{SK}_0^* = \perp$: return \perp</p> <p>22 $\pi_P^t.\text{test} := \text{true}$</p> <p>23 return SK_β^*</p> <p>Oracle CORRUPT(U, S)</p> <p>24 if $(U, S) \in \mathcal{C}$: return \perp</p> <p>25 $\mathcal{C} := \mathcal{C} \cup \{(U, S)\}$</p> <p>26 return $\text{pw}_{U,S}$</p> <p>Oracle E_1(pw, pk)</p> <p>27 if $\exists (\text{pw}, \text{pk}, e_1, *) \in \mathcal{L}_1$: return e_1</p> <p>28 $e_1 \xleftarrow{\\$} \mathcal{E}_1 \setminus \mathcal{T}_1, \mathcal{L}_1 := \mathcal{L}_1 \cup \{e_1\}$</p> <p>29 $\mathcal{L}_1 := \mathcal{L}_1 \cup (\text{pw}, \text{pk}, e_1, \text{enc})$</p> <p>30 return e_1</p> <p>Oracle E_2(pw, c)</p> <p>31 if $\exists (\text{pw}, c, e_2, *) \in \mathcal{L}_2$: return e_2</p> <p>32 $e_2 \xleftarrow{\\$} \mathcal{E}_2 \setminus \mathcal{T}_2, \mathcal{L}_2 := \mathcal{L}_2 \cup \{e_2\}$</p> <p>33 $\mathcal{L}_2 := \mathcal{L}_2 \cup (\text{pw}, c, e_2, \text{enc})$</p> <p>34 return e_2</p> <p>Oracle D_1(pw, e_1)</p> <p>35 if $\exists (\text{pw}, \text{pk}, e_1, *) \in \mathcal{L}_1$: return pk</p> <p>36 $\text{pk} \xleftarrow{\\$} \mathcal{PK}, \mathcal{L}_1 := \mathcal{L}_1 \cup (\text{pw}, \text{pk}, e_1, \text{dec})$</p> <p>37 return pk</p> <p>Oracle D_2(pw, e_2)</p> <p>38 if $\exists (\text{pw}, c, e_2, *) \in \mathcal{L}_2$: return c</p> <p>39 $c \xleftarrow{\\$} \mathcal{C}, \mathcal{L}_2 := \mathcal{L}_2 \cup (\text{pw}, c, e_2, \text{dec})$</p> <p>40 return c</p>	<p>Oracle EXECUTE(U, t_1, S, t_2)</p> <p>41 if $\pi_U^{t_1} \neq \perp$ or $\pi_S^{t_2} \neq \perp$</p> <p>42 return \perp</p> <p>43 let $\text{pw} := \text{pw}_{U,S}$</p> <p>44 $(\text{pk}, \text{sk}) \leftarrow \text{KG}(\text{par}), e_1 := E_1(\text{pw}, \text{pk})$</p> <p>45 $(c, k) \leftarrow \text{Encaps}(\text{pk}), e_2 := E_2(\text{pw}, c)$</p> <p>46 $\text{ctxt} := (U, S, e_1, e_2)$</p> <p>47 $\text{SK} := H(\text{ctxt}, \text{pk}, c, k, \text{pw})$</p> <p>48 $\pi_U^{t_1} := ((\text{pk}, \text{sk}, e_1), \text{ctxt}, \text{SK}, \text{true})$</p> <p>49 $\pi_S^{t_2} := ((c, k, e_2), \text{ctxt}, \text{SK}, \text{true})$</p> <p>50 $(\pi_U^{t_1}.\text{fr}, \pi_S^{t_2}.\text{fr}) := (\text{true}, \text{true})$ // G_1</p> <p>51 return (U, e_1, S, e_2)</p> <p>Oracle SENDINIT(U, t_1, S)</p> <p>52 if $\pi_U^{t_1} \neq \perp$: return \perp</p> <p>53 $(\text{pk}, \text{sk}) \leftarrow \text{KG}(\text{par})$</p> <p>54 $e_1 := E_1(\text{pw}_{U,S}, \text{pk})$</p> <p>55 $\pi_U^{t_1} := ((\text{pk}, \text{sk}, e_1), (U, S, e_1, \perp), \perp, \perp)$</p> <p>56 $\pi_U^{t_1}.\text{fr} := \text{false}$ // G_1</p> <p>57 return (U, e_1)</p> <p>Oracle SENDRESP(S, t_2, U, e_1)</p> <p>58 $\pi_S^{t_2} \neq \perp$: return \perp</p> <p>59 if $(U, S) \in \mathcal{C}$: $\pi_S^{t_2}.\text{fr} := \text{false}$ // G_1</p> <p>60 else $\pi_S^{t_2}.\text{fr} := \text{true}$ // G_1</p> <p>61 $\text{pk} := D_1(\text{pw}_{U,S}, e_1)$</p> <p>62 $(c, k) \leftarrow \text{Encaps}(\text{pk})$</p> <p>63 $e_2 := E_2(\text{pw}_{U,S}, c)$</p> <p>64 $\text{ctxt} := (U, S, e_1, e_2)$</p> <p>65 $\text{SK} := H(\text{ctxt}, \text{pk}, c, k, \text{pw}_{U,S})$</p> <p>66 $\pi_S^{t_2} := ((c, k, e_2), \text{ctxt}, \text{SK}, \text{true})$</p> <p>67 return (S, e_2)</p> <p>Oracle SENDTERINIT(U, t_1, S, e_2)</p> <p>68 if $\pi_U^{t_1} = \perp$ and $\pi_U^{t_1}.\text{tr} \neq (U, S, *, *)$</p> <p>69 return \perp</p> <p>70 let $(\text{pk}, \text{sk}, e_1) := \pi_U^{t_1}.\text{e}$</p> <p>71 $c := D_2(\text{pw}, e_2), k := \text{Decaps}(\text{sk}, c)$</p> <p>72 if $\exists t_2$ s.t. $\pi_S^{t_2}.\text{fr} = \text{true}$ // G_1</p> <p>73 and $\pi_S^{t_2}.\text{tr} = (U, S, e_1, e_2)$ // G_1</p> <p>74 $\pi_U^{t_1}.\text{fr} := \text{true}$ // G_1</p> <p>75 else if $(U, S) \notin \mathcal{C}$: $\pi_U^{t_1}.\text{fr} := \text{true}$ // G_1</p> <p>76 else $\pi_U^{t_1}.\text{fr} := \text{false}$ // G_1</p> <p>77 $\text{ctxt} := (U, S, e_1, e_2)$</p> <p>78 $\text{SK} := H(\text{ctxt}, \text{pk}, c, k, \text{pw}_{U,S})$</p> <p>79 $\pi_U^{t_1}.\text{(tr, key, acc)} := (\text{ctxt}, \text{SK}, \text{true})$</p> <p>80 return true</p> <p>Oracle H$(U, S, e_1, e_2, \text{pk}, c, k, \text{pw})$</p> <p>81 if $\mathcal{L}_H[U, S, e_1, e_2, \text{pk}, c, k, \text{pw}] = \perp$</p> <p>82 $\text{SK} \xleftarrow{\\$} \mathcal{SK}$</p> <p>83 $\mathcal{L}_H[U, S, e_1, e_2, \text{pk}, c, k, \text{pw}] := \text{SK}$</p> <p>84 return $\mathcal{L}_H[U, S, e_1, e_2, \text{pk}, c, k, \text{pw}]$</p>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Figure 8: Games in proving Theorem 4.1. \mathcal{A} has access to the set of PAKE oracles $\{\text{EXECUTE}, \text{SENDINIT}, \text{SENDRESP}, \text{SENDTERINIT}, \text{CORRUPT}, \text{REVEAL}, \text{TEST}\}$, random oracle H , and ideal ciphers $\text{IC}_1 = (E_1, D_1)$ and $\text{IC}_2 = (E_2, D_2)$.

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 4.1, 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 IC₁, IC₂, and H, respectively. By birthday bounds and union bounds, such collision events happen within probability $S^2(\eta_{pk} + \eta_{ct}) + \frac{(q_1^2 + S^2)}{|\mathcal{E}_1|} + \frac{(q_2^2 + S^2)}{|\mathcal{E}_2|} + \frac{q_1^2}{|\mathcal{PK}|} + \frac{q_2^2}{|\mathcal{C}|} + \frac{(q_H^2 + S^2)}{|\mathcal{SK}|}$. Game \mathbf{G}_0 is the same as BPR_{PAKE} except that we define such collision events in \mathbf{G}_0 , we have

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

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 Freshness procedure in the TEST oracle, we assign an additional variable \mathbf{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 $\pi.\mathbf{fr}$ in real time according to the freshness definition (cf. Definition 3.3). This change is conceptual, so we have

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

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.

Oracle EXECUTE(U, t_1, S, t_2)	Oracle $D_1(\text{pw}, e_1)$
01 if $\pi_U^{t_1} \neq \perp$ or $\pi_S^{t_2} \neq \perp$	18 if $\exists (\text{pw}, \text{pk}, e_1, *) \in \mathcal{L}_1$
02 return \perp	19 return pk
03 $\text{pw} := \text{pw}_{U,S}$	20 $\text{pk} \xleftarrow{\$} \mathcal{PK}$ // \mathbf{G}_1
04 $(\text{pk}, \text{sk}) \leftarrow \text{KG}(\text{par}), e_1 := E_1(\text{pw}, \text{pk})$ // $\mathbf{G}_1\text{-}\mathbf{G}_4$	21 $(\text{pk}, \text{sk}) \leftarrow \text{KG}$ // $\mathbf{G}_2\text{-}\mathbf{G}_5$
05 $(\text{c}, \text{k}) \leftarrow \text{Encaps}(\text{pk}), e_2 := E_2(\text{pw}, \text{c})$ // $\mathbf{G}_1\text{-}\mathbf{G}_3$	22 $\mathcal{L}_{\text{key}} := \mathcal{L}_{\text{key}} \cup \{(\text{pk}, \text{sk})\}$ // $\mathbf{G}_2\text{-}\mathbf{G}_5$
06 $\text{c} \xleftarrow{\$} \mathcal{C}, e_2 := E_2(\text{pw}, \text{c})$ // \mathbf{G}_4	23 $\mathcal{L}_1 := \mathcal{L}_1 \cup \{(\text{pw}, \text{pk}, e_1, \text{dec})\}$
07 $e_1 \xleftarrow{\$} \mathcal{E}_1 \setminus \mathcal{T}_1, \mathcal{T}_1 := \mathcal{T}_1 \cup \{e_1\}$ // \mathbf{G}_5	24 return pk
08 $e_2 \xleftarrow{\$} \mathcal{E}_2 \setminus \mathcal{T}_2, \mathcal{T}_2 := \mathcal{T}_2 \cup \{e_2\}$ // \mathbf{G}_5	
09 $\text{ctxt} := (U, S, e_1, e_2)$	
10 $\text{SK} := \text{H}(\text{ctxt}, \text{pk}, \text{c}, \text{k}, \text{pw})$ // $\mathbf{G}_1\text{-}\mathbf{G}_2$	
11 $\text{SK} \xleftarrow{\$} \mathcal{SK}$ // $\mathbf{G}_3\text{-}\mathbf{G}_5$	
12 $\pi_U^{t_1} := ((\text{pk}, \text{sk}, e_1), \text{ctxt}, \text{SK}, \text{true})$ // $\mathbf{G}_1\text{-}\mathbf{G}_3$	
13 $\pi_S^{t_2} := ((\text{c}, \text{k}, e_2), \text{ctxt}, \text{SK}, \text{true})$ // $\mathbf{G}_1\text{-}\mathbf{G}_3$	
14 $\pi_U^{t_1} := ((\perp, \perp, e_1), \text{ctxt}, \text{SK}, \text{true})$ // $\mathbf{G}_4\text{-}\mathbf{G}_5$	
15 $\pi_S^{t_2} := ((\perp, \perp, e_2), \text{ctxt}, \text{SK}, \text{true})$ // $\mathbf{G}_4\text{-}\mathbf{G}_5$	
16 $(\pi_U^{t_1}.\mathbf{fr}, \pi_S^{t_2}.\mathbf{fr}) := (\text{true}, \text{true})$	
17 return (U, e_1, S, e_2)	

Figure 9: Oracles EXECUTE and D_1 in the games sequence $\mathbf{G}_1\text{-}\mathbf{G}_5$.

Game \mathbf{G}_2 . We change the output of D_1 . When \mathcal{A} queries $D_1(\text{pw}, e_1)$ where e_1 is not generated from $E_1(\text{pw}, \cdot)$, we generate pk via $(\text{pk}, \text{sk}) \leftarrow \text{KG}$ instead of $\text{pk} \xleftarrow{\$} \mathcal{PK}$. Such (pk, sk) is recorded in \mathcal{L}_{key} . cf. Lines 20 to 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 4.4. For readability, we continue the proof of Theorem 4.1 and postpone the proof of Lemma 4.4 to Supp. Mat. A.

Lemma 4.4 *With notations and assumptions from \mathbf{G}_1 and \mathbf{G}_2 in the proof of Theorem 4.1, there is an adversary \mathcal{B}_1 with $\mathbf{T}(\mathcal{B}_1) \approx \mathbf{T}(\mathcal{A})$ and*

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

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

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

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

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

We construct an adversary \mathcal{B}_2 against the OW-PCA security of KEM in Figure 10 such that $\mathbf{T}(\mathcal{B}_2) \approx \mathbf{T}(\mathcal{A})$ and $\Pr[\text{Query}_{\text{exec}}] \leq \text{Adv}_{\text{KEM}}^{(S,1)\text{-OW-PCA}}(\mathcal{B}_2)$. Concretely, \mathcal{B}_2 inputs a OW-PCA challenge $(\text{par}, \mathbf{pk}, \mathbf{c})$ 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 $\mathcal{B}_2^{\text{Pco}(\cdot, \cdot)}(\text{par}, \mathbf{pk}, \mathbf{c})$	Oracle EXECUTE(\mathbf{U}, t_1, S, t_2)
01 $\text{cnt} := 0, \mathcal{L}_E := \emptyset$	18 if $\pi_U^{t_1} \neq \perp$ or $\pi_S^{t_2} \neq \perp$
02 $i^* := \perp, j^* := \perp, k^* := \perp$	19 return \perp
03 $\text{Query}_{\text{exec}} := \text{false}$	20 $\mathbf{pw} := \text{pw}_{\mathbf{U}, S}, \text{cnt} := \text{cnt} + 1$
04 for $(\mathbf{U}, S) \in \mathcal{U} \times \mathcal{S}$	21 $\mathbf{pk} := \mathbf{pk}[\text{cnt}], e_1 := E_1(\mathbf{pw}, \mathbf{pk})$
05 $\text{pw}_{\mathbf{U}, S} \leftarrow \mathcal{PW}$	22 $\mathbf{c} := \mathbf{c}[\text{cnt}, 1], e_2 := E_2(\mathbf{pw}, \mathbf{c})$
06 $\mathcal{C} := \emptyset, \beta \leftarrow \{0, 1\}$	23 $\text{ctxt} := (\mathbf{U}, S, e_1, e_2)$
07 $b' \leftarrow \mathcal{A}^{O, \mathbf{H}, \text{IC}_1, \text{IC}_2}(\text{par})$	24 $\mathcal{L}_E := \mathcal{L}_E \cup \{(\text{ctxt}, (\mathbf{pk}, \text{cnt}), \mathbf{c}, \mathbf{pw})\}$
08 return (i^*, j^*, k^*)	25 $\text{SK} \xleftarrow{\$} \mathcal{SK}$
Oracle $\mathbf{H}(\mathbf{U}, S, e_1, e_2, \mathbf{pk}, \mathbf{c}, k, \mathbf{pw})$	26 $\pi_U^{t_1} := ((\mathbf{pk}, \perp, e_1), \text{tr}, \text{SK}, \text{true})$
09 $\text{ctxt} := (\mathbf{U}, S, e_1, e_2)$	27 $\pi_S^{t_2} := ((\mathbf{c}, \perp, e_2), \text{tr}, \text{SK}, \text{true})$
10 if $\exists i'$ s.t. $(\text{ctxt}, (\mathbf{pk}, i'), \mathbf{c}, \mathbf{pw}) \in \mathcal{L}_E$	28 $(\pi_U^{t_1}.\text{fr}, \pi_S^{t_2}.\text{fr}) := (\text{true}, \text{true})$
11 and $\text{PCO}(\text{cnt}^*, \mathbf{c}, k) = 1$	29 return $(\mathbf{U}, e_1, S, e_2)$
12 $\text{Query}_{\text{exec}} := \text{true}$	
13 $(i^*, j^*, k^*) := (i', 1, k)$	
14 if $\mathcal{L}_H[\mathbf{U}, S, e_1, e_2, \mathbf{pk}, \mathbf{c}, k, \mathbf{pw}] = \perp$	
15 $\text{SK} \xleftarrow{\$} \mathcal{SK}$	
16 $\mathcal{L}_H[\mathbf{U}, S, e_1, e_2, \mathbf{pk}, \mathbf{c}, k, \mathbf{pw}] := \text{SK}$	
17 return $\mathcal{L}_H[\mathbf{U}, S, e_1, e_2, \mathbf{pk}, \mathbf{c}, k, \mathbf{pw}]$	

Figure 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 \mathbf{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}_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}_E and PCO to determine whether $\text{Query}_{\text{exec}}$ happens (cf. Lines 10 to 13).

If \mathcal{A} queried $\mathbf{H}(\mathbf{U}, S, e_1, e_2, \mathbf{pk}, \mathbf{c}, k, \mathbf{pw})$, where $(\mathbf{U}, S, e_1, e_2, \mathbf{pk}, \mathbf{c}, k, \mathbf{pw})$ 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 $\text{cnt}^* \in [S]$ such that $(\mathbf{U}, S, e_1, e_2, (\mathbf{pk}, \text{cnt}^*), \mathbf{c}, \mathbf{pw}) \in \mathcal{L}_E$, $\mathbf{c} = \mathbf{c}[\text{cnt}^*, 1]$, and $k = \text{Decaps}(\text{sk}, \mathbf{c})$, where sk is the secret key of $\mathbf{pk}[\text{cnt}^*]$. This means that k is the OW solution of $\mathbf{c}[\text{cnt}^*, 1]$, and thus \mathcal{B}_2 records the OW solution (cf. Line 13) and returns it when the game ends. Therefore, we have

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

Game \mathbf{G}_4 . We change the generation of \mathbf{c} in EXECUTE (cf. Line 06). In this game, \mathbf{c} is sampled from \mathcal{C} uniformly at random instead of using Encaps. Moreover, we no longer store the information about \mathbf{pk} , sk , \mathbf{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 4.5. We continue the proof of Theorem 4.1 and postpone the proof of Lemma 4.5 to Supp. Mat. A.

Lemma 4.5 *With notations and assumptions from \mathbf{G}_3 and \mathbf{G}_4 in the proof of Theorem 4.1, there is an adversary \mathcal{B}_3 with $\mathbf{T}(\mathcal{B}_3) \approx \mathbf{T}(\mathcal{A})$ and*

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

Game \mathbf{G}_5 . We postpone the generation of pk and c in EXECUTE. Concretely, when \mathcal{A} issues a query (U, t_1, 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 $D_1(\text{pw}_{U,S}, e_1)$ or $D_2(\text{pw}_{U,S}, e_2)$, respectively. The change made in \mathbf{G}_2 ensures that pk output by $D_1(\text{pw}_{U,S}, e_1)$ is generated using KG , and the change made in \mathbf{G}_4 ensures that c output by $D_2(\text{pw}_{U,S}, e_2)$ is generated via uniformly sampling over \mathcal{C} . Therefore, \mathbf{G}_5 is conceptually equivalent to \mathbf{G}_4 , which means

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

Game \mathbf{G}_6 . We rewrite the codes of SENDINIT, SENDRESP, and SENDTERINIT in Figure 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 $\pi_U^{t_1}$ 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 $\pi_U^{t_1}$'s session key without decapsulation. There are at most S queries to SENDTERINIT, by a union bound, we have

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

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

- When \mathcal{A} queries SENDRESP(S, t_2, U, e_1), if (U, S) is uncorrupted, e_1 is not generated from U 's instance (cf. Line 37), and $\exists \text{pk}$ such that e_1 is generated via querying $E_1(\text{pw}_{U,S}, \text{pk})$, then we set $\text{Guess}_{\text{ser}}$ as **true** (cf. Lines 23 to 24).
- When \mathcal{A} queries SENDTERINIT(U, t_1, S, e_2), if $\pi_U^{t_1}$ does not have matching session, (U, S) is uncorrupted, e_2 is not generated from S 's instance (cf. Line 30), and $\exists c$ such that e_2 is generated via querying $E_2(\text{pw}_{U,S}, c)$, then we set $\text{Guess}_{\text{user}}$ as **true** (cf. Lines 53 to 54).

These two flags are internal and do not influence the game, and thus \mathbf{G}_7 is equivalent to \mathbf{G}_6 .

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

This step is crucial for our proof. Looking ahead, \mathcal{A} triggered $\text{Guess}_{\text{user}}$ (or $\text{Guess}_{\text{ser}}$, similarly) means that \mathcal{A} queried $E_1(\text{pw}_{U,S}, \text{pk})$ for some pk without corrupting $\text{pw}_{U,S}$. 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 $\text{Guess}_{\text{user}}$ will generate uniformly random session keys. Concretely, when \mathcal{A} queries SENDTERINIT(U, t_1, S, e_2), if $\pi_U^{t_1}$ does not have matching instance, (U, S) is uncorrupted, and e_2 does not trigger $\text{Guess}_{\text{user}}$, then we sample the session key uniformly at random and independent of H (cf. Lines 55 to 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 $\text{Query}_{\text{send}}$ be such querying event. To bound the happening probability of $\text{Query}_{\text{send}}$, we construct an reduction \mathcal{B}_4 with $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B}_4)$ in Figure 12 which attacks OW-rPCA security of KEM. \mathcal{B}_4 works as follows:

<p>Game G_6-G_{10}</p> <p>01 $\text{par} \leftarrow \text{Setup}$</p> <p>02 for $(U, S) \in \mathcal{U}$: $\text{pw}_{U,S} \leftarrow \mathcal{PW}$</p> <p>03 $\mathcal{C} := \emptyset, \beta \leftarrow \{0, 1\}$</p> <p>04 $\text{Guess}_{\text{user}} := \text{false}$</p> <p>05 $\text{Guess}_{\text{ser}} := \text{false}$</p> <p>06 $b' \leftarrow \mathcal{A}^{O, H, \mathcal{L}_1, \mathcal{L}_2}(\text{par})$</p> <p>07 return $\beta == b'$</p> <p>Oracle SENDRESP(S, t_2, U, e_1)</p> <p>08 $\pi_S^{t_2} \neq \perp$: return \perp</p> <p>09 if $(U, S) \in \mathcal{C}$</p> <p>10 $\pi_S^{t_2}.\text{fr} := \text{false}$</p> <p>11 $\text{pk} := D_1(\text{pw}_{U,S}, e_1)$</p> <p>12 $(c, k) \leftarrow \text{Encaps}(\text{pk})$</p> <p>13 $e_2 := E_2(\text{pw}_{U,S}, c)$</p> <p>14 $\text{ctxt} := (U, S, e_1, e_2)$</p> <p>15 $\text{SK} := H(\text{ctxt}, \text{pk}, c, k, \text{pw}_{U,S})$</p> <p>16 else</p> <p>17 $\pi_S^{t_2}.\text{fr} := \text{true}$</p> <p>18 $\text{pk} := D_1(\text{pw}_{U,S}, e_1)$</p> <p>19 $(c, k) \leftarrow \text{Encaps}(\text{pk})$</p> <p>20 $e_2 := E_2(\text{pw}_{U,S}, c)$</p> <p>21 $\text{ctxt} := (U, S, e_1, e_2)$</p> <p>22 $\text{SK} := H(\text{ctxt}, \text{pk}, c, k, \text{pw}_{U,S})$</p> <p>23 if $e_1 \notin \mathcal{L}_1^U$ and $\exists \text{pk s.t.}$ $(\text{pw}_{U,S}, \text{pk}, e_1, \text{enc}) \in \mathcal{L}_1$</p> <p>24 $\text{Guess}_{\text{ser}} := \text{true}$</p> <p>25 else</p> <p>26 $\text{SK} \stackrel{\\$}{\leftarrow} \mathcal{SK}$</p> <p>27 $c \stackrel{\\$}{\leftarrow} \mathcal{C}, e_2 := E_2(\text{pw}_{U,S}, c)$</p> <p>28 $\pi_S^{t_2}.\text{key}, \text{tr} := ((c, k, e_2), \text{ctxt})$</p> <p>29 $\pi_S^{t_2}.\text{acc} := (\text{SK}, \text{true})$</p> <p>30 $\mathcal{L}_2^S := \mathcal{L}_2^S \cup \{e_2\}$</p> <p>31 return (S, e_2)</p>	<p>Oracle SENDINIT(U, t_1, S)</p> <p>32 if $\pi_U^{t_1} \neq \perp$: return \perp</p> <p>33 $(\text{pk}, \text{sk}) \leftarrow \text{KG}(\text{par}), e_1 := E_1(\text{pw}_{U,S}, \text{pk})$</p> <p>34 $e_1 \stackrel{\\$}{\leftarrow} \mathcal{E}_1 \setminus \mathcal{T}_1, \mathcal{T}_1 := \mathcal{T}_1 \cup \{e_1\}$</p> <p>35 $\text{pk} := D_1(\text{pw}_{U,S}, e_1)$</p> <p>36 Retrieve $\text{sk s.t.} (\text{pk}, \text{sk}) \in \mathcal{L}_{\text{key}}$</p> <p>37 $\mathcal{L}_1^U := \mathcal{L}_1^U \cup \{e_1\}$</p> <p>38 $\pi_U^{t_1} := ((\text{pk}, \text{sk}, e_1), (U, S, e_1, \perp), \perp, \perp)$</p> <p>39 $\pi_U^{t_1}.\text{fr} := \text{false}$</p> <p>40 return (U, e_1)</p> <p>Oracle SENDTERINIT(U, t_1, S, e_2)</p> <p>41 if $\pi_U^{t_1} = \perp$ and $\pi_U^{t_1}.\text{tr} \neq (U, S, *, *)$</p> <p>42 return \perp</p> <p>43 $(\text{pk}, \text{sk}, e_1) := \pi_U^{t_1}.\text{e}$</p> <p>44 if $\exists t_2$ s.t. $\pi_S^{t_2}.\text{fr} = \text{true}$</p> <p>45 and $\pi_S^{t_2}.\text{tr} = (U, S, e_1, e_2)$</p> <p>46 $\pi_U^{t_1}.\text{fr} := \text{true}, \text{SK} := \pi_S^{t_2}.\text{key}$</p> <p>47 else</p> <p>48 $\text{ctxt} := (U, S, e_1, e_2)$</p> <p>49 if $(U, S) \notin \mathcal{C}$</p> <p>50 $\pi_U^{t_1}.\text{fr} := \text{true}$</p> <p>51 $c := D_2(\text{pw}_{U,S}, e_2), k := \text{Decaps}(\text{sk}, c)$</p> <p>52 $\text{SK} := H(\text{ctxt}, \text{pk}, c, k, \text{pw}_{U,S})$</p> <p>53 if $e_2 \notin \mathcal{L}_2^S$ and $\exists c$ s.t. $(\text{pw}_{U,S}, c, e_2, \text{enc}) \in \mathcal{L}_2$</p> <p>54 $\text{Guess}_{\text{user}} := \text{true}$</p> <p>55 else</p> <p>56 $\text{SK} \stackrel{\\$}{\leftarrow} \mathcal{SK}$</p> <p>57 else</p> <p>58 $\pi_U^{t_1}.\text{fr} := \text{false}$</p> <p>59 $c := D_2(\text{pw}_{U,S}, e_2), k := \text{Decaps}(\text{sk}, c)$</p> <p>60 $\text{SK} := H(\text{ctxt}, \text{pk}, c, k, \text{pw}_{U,S})$</p> <p>61 $\pi_U^{t_1}.\text{tr}, \text{key}, \text{acc} := (\text{ctxt}, \text{SK}, \text{true})$</p> <p>62 return true</p>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Figure 11: Oracles SENDINIT, SENDRESP, and SENDTERINIT in games G_6 - G_{10} . For any user U , \mathcal{L}_1^U records all e_1 sent by U . Similarly, \mathcal{L}_2^S records all e_2 sent by server S . All these lists are initialized as \emptyset .

1. On input a OW-rPCA challenge $(\text{par}, \text{pk}, \text{c})$, \mathcal{B}_4 embeds public keys in 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}'_{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 (U, t_1, S, e_2) , where $\pi_U^{t_1}$ does not have fresh matching instance and either e_2 triggers $\text{Guess}_{\text{user}}$ 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 (U, t_1, S, e_2) , if $\pi_U^{t_1}$ does not have fresh matching instance, (U, S) is corrupted, and e_2 does not trigger $\text{Guess}_{\text{user}}$, then e_2 is not generated by querying $E_2(\text{pw}_{U,S}, e_2)$, which means that $c = D_2(\text{pw}_{U,S}, e_2)$ is one of the embedded ciphertext (cf. Line 15). \mathcal{B}_4 records such query in \mathcal{L}_{SK} (cf. Line 46) to determine whether $\text{Query}_{\text{send}}$ happens.

<p>Reduction $\mathcal{B}_4^{\text{PCO}}(\text{par}, \text{pk}, \text{c})$</p> <pre> 01 cnt₁ := 0, cnt₂ := 0, \mathcal{L}_{ct} := \emptyset 02 $i^* := \perp, j^* := \perp, k^* := \perp$ 03 for $(U, S) \in \mathcal{U}$: $\text{pw}_{U,S} \leftarrow \mathcal{PW}$ 04 $\mathcal{C} := \emptyset, \beta \leftarrow \{0, 1\}$ 05 $\text{Guess}_{\text{user}} := \text{false}, \text{Guess}_{\text{ser}} := \text{false}$ 06 $\text{Query}_{\text{send}} := \text{false}$ 07 $b' \leftarrow \mathcal{A}^{O, H, IC_1, IC_2}(\text{par})$ 08 return (i^*, j^*, k^*) </pre> <p>Oracle $\text{SENDINIT}(U, t_1, S)$</p> <pre> 09 if $\pi_U^{t_1} \neq \perp$: return \perp 10 cnt₁ := cnt₁ + 1, $\text{pk} := \text{pk}[\text{cnt}_1]$ 11 $e_1 := E_1(\text{pw}_{U,S}, \text{pk}), \mathcal{L}_1^U := \mathcal{L}_1^U \cup \{e_1\}$ 12 $\pi_U^{t_1} := ((\text{pk}, \text{cnt}_1, e_1), (U, S, e_1, \perp), \perp, \perp)$ 13 return (U, e_1) </pre> <p>Oracle $D_2(\text{pw}, e_2)$</p> <pre> 14 if $\exists (\text{pw}, \text{c}, e_2, *) \in \mathcal{L}_2$: return c 15 cnt₂ := cnt₂ + 1, $\text{c} := \text{c}[\text{cnt}_2]$ 16 $\mathcal{L}_{\text{ct}} := \mathcal{L}_{\text{ct}} \cup \{(\text{c}, \text{cnt}_2)\}$ 17 $\mathcal{L}_2 := \mathcal{L}_2 \cup (\text{pw}, \text{c}, e_2, \text{dec})$ 18 return c </pre> <p>Oracle $H(U, S, e_1, e_2, \text{pk}, \text{c}, \text{k}, \text{pw})$</p> <pre> 19 $\text{ctxt} := (U, S, e_1, e_2)$ 20 if $\exists i, \text{SK}$ s.t. $(\text{ctxt}, (\text{pk}, i), \text{c}, \text{pw}, \text{SK}) \in \mathcal{L}'_{\text{SK}}$ 21 and $\text{PCO}(i, \text{c}, \text{k}) = 1$ 22 $\mathcal{L}_H[U, S, e_1, e_2, \text{pk}, \text{c}, \text{k}, \text{pw}] := \text{SK}$ 23 if $\exists i, j$ s.t. $(\text{ctxt}, (\text{pk}, i), (\text{c}, j)) \in \mathcal{L}_{\text{SK}}$ 24 and $\text{PCO}(i, \text{c}, \text{k}) = 1$ 25 $(i^*, j^*, k^*) := (i, j, \text{k}), \text{Query}_{\text{send}} := \text{true}$ 26 if $\mathcal{L}_H[U, S, e_1, e_2, \text{pk}, \text{c}, \text{k}, \text{pw}] = \perp$ 27 $\mathcal{L}_H[U, S, e_1, e_2, \text{pk}, \text{c}, \text{k}, \text{pw}] := \text{SK} \stackrel{\\$}{\leftarrow} \text{SK}$ 28 return $\mathcal{L}_H[U, S, e_1, e_2, \text{pk}, \text{c}, \text{k}, \text{pw}]$ </pre>	<p>Oracle $\text{SENDTERINIT}(U, t_1, S, e_2)$</p> <pre> 29 if $\pi_U^{t_1} = \perp$ and $\pi_U^{t_1}.\text{tr} \neq (U, S, *, *)$ 30 return \perp 31 $(\text{pk}, i, e_1) := \pi_U^{t_1}.\text{e}$ 32 if $\exists t_2$ s.t. $\pi_S^{t_2}.\text{fr} = \text{true}$ 33 and $\pi_S^{t_2}.\text{tr} = (U, S, e_1, e_2)$ 34 $\pi_U^{t_1}.\text{fr} := \text{true}, \text{SK} := \pi_S^{t_2}.\text{key}$ 35 else 36 $\text{ctxt} := (U, S, e_1, e_2), \text{c} := D_2(\text{pw}, e_2)$ 37 if $(U, S) \notin \mathcal{C}$ 38 $\pi_U^{t_1}.\text{fr} := \text{true}$ 39 $\text{c} := D_2(\text{pw}, e_2)$ 40 if $e_2 \notin \mathcal{L}_2^S$ and $\exists \text{c}$ s.t. 41 $(\text{pw}_{U,S}, \text{c}, e_2, \text{enc}) \in \mathcal{L}_2$ 42 $\text{Guess}_{\text{user}} := \text{true}$ 43 $\text{SK} := \text{Patch}(\text{ctxt}, \text{pk}, i, \text{c})$ 44 else 45 Retrieve j s.t. $(\text{c}, j) \in \mathcal{L}_2$ 46 $\text{SK} \stackrel{\\$}{\leftarrow} \text{SK}$ 47 $\mathcal{L}_{\text{SK}} := \mathcal{L}_{\text{SK}} \cup (\text{ctxt}, (\text{pk}, i), (\text{c}, j))$ 48 else 49 $\pi_U^{t_1}.\text{fr} := \text{false}$ 50 $\text{SK} := \text{Patch}(\text{ctxt}, \text{pk}, i, \text{c})$ 51 $\pi_U^{t_1}.\text{tr} := (\text{tr}, \text{key}, \text{acc}) := (\text{ctxt}, \text{SK}, \text{true})$ 52 return true </pre> <p>Procedure $\text{Patch}(\text{ctxt}, \text{pk}, i, \text{c})$</p> <pre> 52 $(U, S, e_1, e_2) := \text{ctxt}, \text{pw} := \text{pw}_{U,S}$ 53 if $\exists \text{k}$ s.t. $\text{PCO}(i, \text{k}, \text{c}) = 1$ 54 and $\mathcal{L}_H[\text{ctxt}, \text{pk}, \text{c}, \text{k}, \text{pw}] \neq \perp$ 55 $\text{SK} := \mathcal{L}_H[\text{ctxt}, \text{pk}, \text{c}, \text{k}, \text{pw}]$ 56 else 57 $\text{SK} \stackrel{\\$}{\leftarrow} \text{SK}$ 58 $\mathcal{L}'_{\text{SK}} := \mathcal{L}'_{\text{SK}} \cup (\text{ctxt}, (\text{pk}, i), \text{c}, \text{pw}, \text{SK})$ 59 return SK </pre>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Figure 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. List \mathcal{L}'_{SK} is used for recording random session keys that may be patched, and list \mathcal{L}_{SK} is used to determine whether $\text{Query}_{\text{send}}$ happens. All other oracles not shown in the figure are the same as in \mathbf{G}_8 (cf. Figures 8, 9 and 11).

When \mathcal{A} queried $H(U, S, e_1, e_2, \text{pk}, \text{c}, \text{k}, \text{pw}_{U,S})$, if this query match one record in \mathcal{L}_{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, $\text{Query}_{\text{send}}$ 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 $\text{Query}_{\text{send}}$ happens, then \mathcal{B}_4 finds the OW solution of one of the challenge ciphertexts. Therefore, we have

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

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

1. In SENDINIT , instead of generating $(\text{pk}, \text{sk}) \leftarrow \text{KG}$ and $e_1 := E_1(\text{pw}_{U,S}, \text{pk})$, we firstly sample e_1 uniformly at random and then generate (pk, sk) by querying $D_1(\text{pw}_{U,S}, e_1)$ (cf. Lines 34 to 36).
2. Fresh server instances that do not trigger $\text{Guess}_{\text{ser}}$ will generate uniformly random session keys. Concretely, when \mathcal{A} queries $\text{SENDRESP}(S, t_2, U, e_1)$, if (U, S) is uncorrupted and e_1 does not trigger $\text{Guess}_{\text{ser}}$, 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 $\text{Query}_{\text{resp}}$ be such querying event. We construct an reduction \mathcal{B}_5 with $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B}_5)$ in Figure 12 to bound the happening probability of $\text{Query}_{\text{resp}}$. \mathcal{B}_5 attacks OW-PCA security of KEM and works as follows:

<p>Reduction $\mathcal{B}_5(\text{par}, \mathbf{pk}, \mathbf{c})$</p> <pre> 01 $\text{cnt}_1 := 0, i^* := \perp, j^* := \perp, k^* := \perp$ 02 for $(U, S) \in \mathcal{U}$ 03 $\text{pw}_{U,S} \leftarrow \mathcal{PW}, \mathcal{L}_1^U := \emptyset, \mathcal{L}_2^S := \emptyset$ 04 $\mathcal{C} := \emptyset, \beta \leftarrow \{0, 1\}$ 05 $\text{Guess}_{\text{user}} := \text{false}, \text{Guess}_{\text{ser}} := \text{false}$ 06 $\text{Query}_{\text{resp}} := \text{false}$ 07 $b' \leftarrow \mathcal{A}^{O, H, \mathcal{C}_1, \mathcal{C}_2}(\text{par})$ 08 return (i^*, j^*, k^*) </pre> <p>Oracle $\text{SENDTERINIT}(U, t_1, S, e_2)$</p> <pre> 09 if $\pi_U^{t_1} = \perp$ and $\pi_U^{t_1}.\text{tr} \neq (U, S, *, *)$ 10 return \perp 11 $(\mathbf{pk}, i, e_1) := \pi_U^{t_1}.e, c := D_2(\text{pw}, e_2)$ 12 if $\exists t_2$ s.t. $\pi_S^{t_2}.\text{fr} = \text{true}$ 13 and $\pi_S^{t_2}.\text{tr} = (U, S, e_1, e_2)$ 14 $\pi_U^{t_1}.\text{fr} := \text{true}, \text{SK} := \pi_S^{t_2}.\text{key}$ 15 else 16 $\text{ctxt} := (U, S, e_1, e_2)$ 17 if $(U, S) \notin \mathcal{C}$ 18 $\pi_U^{t_1}.\text{fr} := \text{true}$ 19 if $e_2 \notin \mathcal{L}_2^S$ and $\exists c$ s.t. 20 $(\text{pw}_{U,S}, c, e_2, \text{enc}) \in \mathcal{L}_2$ 21 $\text{Guess}_{\text{user}} := \text{true}$ 22 $\text{SK} := \text{Patch}(\text{ctxt}, \mathbf{pk}, i, c)$ 23 else $\text{SK} \stackrel{\\$}{\leftarrow} \mathcal{SK}$ 24 else 25 $\pi_U^{t_1}.\text{fr} := \text{false}$ 26 $\text{SK} := \text{Patch}(\text{ctxt}, \mathbf{pk}, i, c)$ 27 $\pi_U^{t_1}(\text{tr}, \text{key}, \text{acc}) := (\text{ctxt}, \text{SK}, \text{true})$ 28 return true </pre> <p>Oracle $H((U, S, e_1, e_2), \mathbf{pk}, c, k, \text{pw})$</p> <pre> 56 $\text{ctxt} := (U, S, e_1, e_2)$ 57 if $\exists i, \text{SK}$ s.t. $(\text{ctxt}, (\mathbf{pk}, i), c, \text{pw}, \text{SK}) \in \mathcal{L}'_{\text{SK}}$ and $\text{PCO}(i, c, k) = 1$ 58 $\mathcal{L}_H[U, S, e_1, e_2, \mathbf{pk}, c, k, \text{pw}] := \text{SK}$ 59 if $\exists i, j$ s.t. $(\text{ctxt}, (\mathbf{pk}, i), (c, j)) \in \mathcal{L}_{\text{SK}}$ and $\text{PCO}(i, c, k) = 1$ 60 $(i^*, j^*, k^*) := (i, j, k), \text{Query}_{\text{resp}} := \text{true}$ 61 if $\mathcal{L}_H[U, S, e_1, e_2, \mathbf{pk}, c, k, \text{pw}] = \perp$ 62 $\mathcal{L}_H[U, S, e_1, e_2, \mathbf{pk}, c, k, \text{pw}] := \text{SK} \stackrel{\\$}{\leftarrow} \mathcal{SK}$ 63 return $\mathcal{L}_H[U, S, e_1, e_2, \mathbf{pk}, c, k, \text{pw}]$ </pre>	<p>Oracle $D_1(\text{pw}, e_1)$</p> <pre> 28 if $\exists (\text{pw}, \mathbf{pk}, e_1, *) \in \mathcal{L}_1$ 29 return c 30 $\text{cnt}_2[\text{cnt}_1] := 0, \text{cnt}_1 := \text{cnt}_1 + 1$ 31 $\mathbf{pk} := \mathbf{pk}[\text{cnt}_1], \mathcal{L}_{\text{key}} := \mathcal{L}_{\text{key}} \cup \{(\mathbf{pk}, \text{cnt}_1)\}$ 32 $\mathcal{L}_2 := \mathcal{L}_2 \cup (\text{pw}, \mathbf{pk}, e_1, \text{dec})$ 33 return c </pre> <p>Oracle $\text{SENDRESP}(S, t_2, U, e_1)$</p> <pre> 34 $\pi_S^{t_2} \neq \perp$: return \perp 35 $\mathbf{pk} := D_1(\text{pw}_{U,S}, e_1)$ 36 if $(U, S) \in \mathcal{C}$ 37 $\pi_S^{t_2}.\text{fr} := \text{false}$ 38 $(c, k) \leftarrow \text{Encaps}(\mathbf{pk}), e_2 := E_2(\text{pw}_{U,S}, c)$ 39 $\text{ctxt} := (U, S, e_1, e_2)$ 40 $\text{SK} := H(\text{ctxt}, \mathbf{pk}, c, k, \text{pw}_{U,S})$ 41 else 42 $\pi_S^{t_2}.\text{fr} := \text{true}$ 43 if $e_1 \notin \mathcal{L}_1^U$ and $\exists \mathbf{pk}$ s.t. 44 $(\text{pw}_{U,S}, \mathbf{pk}, e_1, \text{enc}) \in \mathcal{L}_1$ 45 $\text{Guess}_{\text{ser}} := \text{true}$ 46 $(c, k) \leftarrow \text{Encaps}(\mathbf{pk}), e_2 := E_2(\text{pw}_{U,S}, c)$ 47 $\text{SK} := H(\text{ctxt}, \mathbf{pk}, c, k, \text{pw}_{U,S})$ 48 else 49 Retrieve i s.t. $(\mathbf{pk}, i) \in \mathcal{L}_{\text{key}}$ 50 $\text{cnt}_2[i] := \text{cnt}_2[i] + 1, j := \text{cnt}_2[i]$ 51 $c := c[i, j], e_2 := E_2(\text{pw}_{U,S}, c)$ 52 $\mathcal{L}_{\text{SK}} := \mathcal{L}_{\text{SK}} \cup \{(\text{ctxt}, (\mathbf{pk}, i), (c, j))\}$ 53 $\text{SK} \stackrel{\\$}{\leftarrow} \mathcal{SK}$ 54 $\mathcal{L}_2^S := \mathcal{L}_2^S \cup \{e_2\}$ 55 $\pi_S^{t_2} := ((c, k, e_2), \text{ctxt}, \text{SK}, \text{true})$ 56 return (S, e_2) </pre>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Figure 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 . List \mathcal{L}'_{SK} is used for recording random session keys that may be patched, and list \mathcal{L}_{SK} is used to determine whether $\text{Query}_{\text{send}}$ happens. All other oracles not shown in the figure are the same as in \mathbf{G}_8 (cf. Figures 8, 9 and 11). Procedure Patch is the same as the one shown in Figure 12.

1. On input a OW-PCA challenge $(\text{par}, \mathbf{pk}, \mathbf{c})$, \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 in SENDINIT are also from \mathbf{pk} . Similar to \mathcal{B}_4 , \mathcal{B}_5 uses the Patch procedure in Figure 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 $\text{SENDRESP}(S, t_2, U, e_1)$, if $\pi_S^{t_2}$ is fresh (which means that (U, S) is uncorrupted) and e_1 does not trigger $\text{Guess}_{\text{ser}}$, then by our definition of $\text{Guess}_{\text{ser}}$, e_1 is not generated by querying $E_1(\text{pw}_{U,S}, \mathbf{pk})$. This means that $\mathbf{pk} = D_1(\text{pw}_{U,S}, e_1)$ is one of the embedded public key (cf. Line 31). In this case, \mathcal{B}_5 embeds one challenge ciphertext with respect to \mathbf{pk} (cf. Line 50) and records such query in \mathcal{L}_{SK} (cf. Line 51) to determine whether $\text{Query}_{\text{resp}}$ happens.

When \mathcal{A} queried $H(U, S, e_1, e_2, \text{pk}, c, k, \text{pw}_{U,S})$, if this query match one record in \mathcal{L}_{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, $\text{Query}_{\text{resp}}$ 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 $(\text{SENDINIT}, \text{SENDRESP})$ 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 $\text{Query}_{\text{resp}}$ happens, then \mathcal{B}_5 finds the OW solution of one of challenge ciphertexts in \mathbf{c} . Therefore, we have

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

Game \mathbf{G}_{10} . We sample KEM ciphertext uniformly at random for server instances that are fresh and do not trigger $\text{Query}_{\text{resp}}$ (cf. Line 27). Similar to the argument of bounding \mathbf{G}_3 and \mathbf{G}_4 (cf. Lemma 4.5), 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 4.6. We continue the proof of Theorem 4.1 and postpone the proof of Lemma 4.6 to Supp. Mat. A.

Lemma 4.6 *With notations and assumptions from \mathbf{G}_9 and \mathbf{G}_{10} in the proof of Theorem 4.1, there is an adversary \mathcal{B}_6 with $\mathbf{T}(\mathcal{B}_6) \approx \mathbf{T}(\mathcal{A})$ and*

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

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

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

- In SENDRESP , if the server instance $\pi_S^{t_2}$ is fresh and does not trigger $\text{Guess}_{\text{ser}}$, then we sample e_2 uniformly at random and without using $\text{pw}_{U,S}$ and c (cf. Lines 33 to 34). Moreover, we only store e_2 as the ephemeral secret of $\pi_S^{t_2}$ (cf. Line 41). These changes are conceptual since we do not need c to compute the session key and if \mathcal{A} queries $D_2(\text{pw}_{U,S}, e_2)$ 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 $\text{pw}_{U,S}$ and pk (cf. Lines 49 to 52) and only store e_1 as the ephemeral secret of $\pi_U^{t_1}$ (cf. Lines 52 to 53 and Line 59). Later, if \mathcal{A} corrupts (U, S) and queries SENDTERINIT to finish the user instance $\pi_U^{t_1}$, we retrieve necessary information to compute the session key (cf. Lines 82 to 83). These changes are also conceptual, since session keys of such instances are independently and uniformly random. We have

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

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

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 and 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 $\text{Guess}_{\text{user}}$ or $\text{Guess}_{\text{ser}}$ 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).

Game G_{10}-G_{12}		Oracle $\text{SENDINIT}(U, t_1, S)$	
01 $\text{par} \leftarrow \text{Setup}$		46 if $\pi_U^{t_1} \neq \perp$: return \perp	
02 for $(U, S) \in \mathcal{U}$: $\text{pw}_{U,S} \leftarrow \mathcal{PW}$	// G_{10}-G_{11}	47 $e_1 \xleftarrow{\$} \mathcal{E}_1 \setminus \mathcal{T}_1, \mathcal{L}_1 := \mathcal{L}_1 \cup \{e_1\}$	
03 $\mathcal{C} := \emptyset, \beta \leftarrow \{0, 1\}$		48 $\mathcal{L}_1^U := \mathcal{L}_1^U \cup \{e_1\}$	
04 $\text{Guess}_{\text{user}} := \text{false}, \text{Guess}_{\text{ser}} := \text{false}$		49 $\text{pk} := D_1(\text{pw}_{U,S}, e_1)$	// G_{10}
05 $b' \leftarrow \mathcal{A}^{O, H, \mathcal{IC}_1, \mathcal{IC}_2}(\text{par})$		50 Retrieve sk s.t. $(\text{pk}, \text{sk}) \in \mathcal{L}_{\text{key}}$	// G_{10}
06 for $(U, S) \in \mathcal{U} \times \mathcal{S}$	// G_{12}	51 $\pi_U^{t_1} := ((\text{pk}, \text{sk}, e_1),$	
07 if $(U, S) \notin \mathcal{C}$: $\text{pw}_{U,S} \leftarrow \mathcal{PW}$	// G_{12}	$(U, S, e_1, \perp), \perp, \perp)$	// G_{10}
08 if $\exists S'$ s.t. $\text{pw}_{U,S'} \in \mathcal{L}_{\text{pw}}$	// G_{12}	52 $\pi_U^{t_1}.e := (\perp, \perp, e_1)$	// G_{11}-G_{12}
09 $\text{Guess}_{\text{user}} := \text{true}$	// G_{12}	53 $\pi_U^{t_1}.\text{tr} := (U, S, e_1, \perp)$	// G_{11}-G_{12}
10 if $\exists U'$ s.t. $\text{pw}_{U',S} \in \mathcal{L}_{\text{pw}}$	// G_{12}	54 $\pi_U^{t_1}.\text{fr} := \text{false}$	
11 $\text{Guess}_{\text{ser}} := \text{true}$	// G_{12}	55 return (U, e_1)	
12 return $\beta == b'$			
Oracle $\text{CORRUPT}(U, S)$		Oracle $\text{SENDTERINIT}(U, t_1, S, e_2)$	
13 if $(U, S) \in \mathcal{C}$: return \perp		56 if $\pi_U^{t_1} = \perp$ and $\pi_U^{t_1}.\text{tr} \neq (U, S, *, *)$	
14 $\mathcal{C} := \mathcal{C} \cup \{(U, S)\}$		57 return \perp	
15 $\text{pw}_{U,S} \leftarrow \mathcal{PW}$	// G_{12}	58 $(\text{pk}, \text{sk}, e_1) := \pi_U^{t_1}.e$	// G_{10}
16 return $\text{pw}_{U,S}$		59 $(\perp, \perp, e_1) := \pi_U^{t_1}.e$	// G_{11}
Oracle $\text{SENDRESP}(S, t_2, U, e_1)$		60 if $\exists t_2$ s.t. $\pi_S^{t_2}.\text{fr} = \text{true}$	
17 $\pi_S^{t_2} \neq \perp$: return \perp		61 and $\pi_S^{t_2}.\text{tr} = (U, S, e_1, e_2)$	
18 if $(U, S) \in \mathcal{C}$		62 $\pi_U^{t_1}.\text{fr} := \text{true}, \text{SK} := \pi_S^{t_2}.\text{key}$	
19 $\pi_S^{t_2}.\text{fr} := \text{false}$		63 else	
20 $\text{pk} := D_1(\text{pw}_{U,S}, e_1), (c, k) \leftarrow \text{Encaps}(\text{pk})$		64 $\text{ctxt} := (U, S, e_1, e_2)$	
21 $e_2 := E_2(\text{pw}_{U,S}, c), \text{ctxt} := (U, S, e_1, e_2)$		65 if $(U, S) \notin \mathcal{C}$	
22 $\text{SK} := H(\text{ctxt}, \text{pk}, c, k, \text{pw}_{U,S})$		66 $\pi_U^{t_1}.\text{fr} := \text{true}$	
23 else		67 if $e_2 \notin \mathcal{L}_2^S$ and $\exists c$ s.t.	
24 $\pi_S^{t_2}.\text{fr} := \text{true}$		$(\text{pw}_{U,S}, c, e_2, \text{enc}) \in \mathcal{L}_2$	// G_{10}-G_{11}
25 if $e_1 \notin \mathcal{L}_1^U$ and $\exists \text{pk}$ s.t.		$\text{pk} := D_1(\text{pw}_{U,S}, e_1)$	// G_{11}
$(\text{pw}_{U,S}, \text{pk}, e_1, \text{enc}) \in \mathcal{L}_1$	// G_{10}-G_{11}	Retrieve sk s.t.	
26 $\text{Guess}_{\text{ser}} := \text{true}$	// G_{10}-G_{11}	$(\text{pk}, \text{sk}) \in \mathcal{L}_{\text{key}}$	// G_{11}
27 $\text{pk} := D_1(\text{pw}_{U,S}, e_1)$	// G_{10}-G_{11}	29 $\text{Guess}_{\text{user}} := \text{true}$	// G_{10}-G_{11}
28 $(c, k) \leftarrow \text{Encaps}(\text{pk})$	// G_{10}-G_{11}	30 $c := D_2(\text{pw}_{U,S}, e_2)$	// G_{10}-G_{11}
29 $e_2 := E_2(\text{pw}_{U,S}, c)$	// G_{10}-G_{11}	31 $k := \text{Decaps}(\text{sk}, c)$	// G_{10}-G_{11}
30 $\text{ctxt} := (U, S, e_1, e_2)$	// G_{10}-G_{11}	32 $\text{SK} := H(\text{ctxt}, \text{pk}, c, k, \text{pw}_{U,S})$	// G_{10}-G_{11}
31 $\text{SK} := H(\text{ctxt}, \text{pk}, c, k, \text{pw}_{U,S})$	// G_{10}-G_{11}	33 else	// G_{10}-G_{11}
32 else	// G_{10}-G_{11}	34 $\text{SK} \xleftarrow{\$} \text{SK}$	// G_{10}-G_{11}
33 $c \leftarrow \mathcal{C}, e_2 := E_2(\text{pw}_{U,S}, c)$	// G_{10}	35 if $e_2 \notin \mathcal{L}_2^S$	
34 $e_2 \xleftarrow{\$} \mathcal{E}_2 \setminus \mathcal{T}_2, \mathcal{T}_2 := \mathcal{T}_2 \cup \{e_2\}$	// G_{11}	36 for (pw, pk) s.t.	
35 $\text{SK} \xleftarrow{\$} \text{SK}$	// G_{10}-G_{11}	$(\text{pw}, \text{pk}, e_1, \text{enc}) \in \mathcal{L}_1$	
36 if $e_1 \notin \mathcal{L}_1^U$	// G_{12}	$\mathcal{L}_{\text{pw}} := \mathcal{L}_{\text{pw}} \cup \{\text{pw}\}$	// G_{12}
37 for (pw, pk) s.t.	// G_{12}	37 $e_2 \xleftarrow{\$} \mathcal{E}_2 \setminus \mathcal{T}_2, \mathcal{T}_2 := \mathcal{T}_2 \cup \{e_2\}$	// G_{12}
$(\text{pw}, \text{pk}, e_1, \text{enc}) \in \mathcal{L}_1$	// G_{12}	38 $\text{SK} \xleftarrow{\$} \text{SK}$	// G_{12}
38 $\mathcal{L}_{\text{pw}} := \mathcal{L}_{\text{pw}} \cup \{\text{pw}\}$	// G_{12}	39 $\pi_S^{t_2}.(e, \text{tr}) := ((c, k, e_2), \text{ctxt})$	// G_{10}
39 $e_2 \xleftarrow{\$} \mathcal{E}_2 \setminus \mathcal{T}_2, \mathcal{T}_2 := \mathcal{T}_2 \cup \{e_2\}$	// G_{12}	40 $\pi_S^{t_2}.(e, \text{tr}) := ((\perp, \perp, e_2), \text{ctxt})$	// G_{11}-G_{12}
40 $\text{SK} \xleftarrow{\$} \text{SK}$	// G_{12}	41 $\pi_S^{t_2}(\text{key}, \text{acc}) := (\text{SK}, \text{true})$	
41 $\pi_S^{t_2}.(e, \text{tr}) := ((c, k, e_2), \text{ctxt})$	// G_{10}	42 $\mathcal{L}_2^S := \mathcal{L}_2^S \cup \{e_2\}$	
42 $\pi_S^{t_2}.(e, \text{tr}) := ((\perp, \perp, e_2), \text{ctxt})$	// G_{11}-G_{12}	43 return (S, e_2)	
43 $\pi_S^{t_2}(\text{key}, \text{acc}) := (\text{SK}, \text{true})$			
44 $\mathcal{L}_2^S := \mathcal{L}_2^S \cup \{e_2\}$			
45 return (S, e_2)			
		74 else	
		75 $\text{SK} \xleftarrow{\$} \text{SK}$	// G_{10}-G_{11}
		76 if $e_2 \notin \mathcal{L}_2^S$	
		77 for (pw, c) s.t.	
		$(\text{pw}, c, e_2, \text{enc}) \in \mathcal{L}_2$	// G_{12}
		$\mathcal{L}_{\text{pw}} := \mathcal{L}_{\text{pw}} \cup \{\text{pw}\}$	// G_{12}
		78 $\text{SK} \xleftarrow{\$} \text{SK}$	// G_{12}
		79 else	
		80 $\pi_U^{t_1}.\text{fr} := \text{false}$	
		81 $\text{pk} := D_1(\text{pw}_{U,S}, e_1)$	// G_{11}-G_{12}
		82 Retrieve sk s.t.	
		$(\text{pk}, \text{sk}) \in \mathcal{L}_{\text{key}}$	// G_{11}-G_{12}
		83 $c := D_2(\text{pw}_{U,S}, e_2)$	
		84 $k := \text{Decaps}(\text{sk}, c)$	
		85 $\text{SK} := H(\text{ctxt}, \text{pk}, c, k, \text{pw}_{U,S})$	
		86 $\pi_U^{t_1}(\text{tr}, \text{key}, \text{acc}) := (\text{ctxt}, \text{SK}, \text{true})$	
		87 return true	
		88 return true	

Figure 14: Oracles SENDINIT , SENDRESP , and SENDTERINIT in games G_{10} - G_{12} .

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

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

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}_{pw} that may trigger GUESS in \mathbf{G}_{11} . If GUESS happens in \mathbf{G}_{12} , then there exists $\text{pw}_{U,S} \in \mathcal{L}_{\text{pw}}$. Moreover, $\text{pw}_{U,S}$ is recorded in \mathcal{L}_{pw} only if (U, S) is uncorrupted. By (cf. Lines 37 to 38 and Lines 76 to 78), $\text{pw}_{U,S} \in \mathcal{L}_{\text{pw}}$ means that there exists (pk, e_1) (resp., (c, e_2)) such that $e_1 \notin \mathcal{L}_1^U$ (resp., $e_2 \notin \mathcal{L}_2^S$) and $(\text{pw}_{U,S}, \text{pk}, e_1, \text{enc}) \in \mathcal{L}_1$ (resp., $(\text{pw}_{U,S}, c, e_2, \text{enc}) \in \mathcal{L}_2$), and thus either $\text{Guess}_{\text{user}}$ or $\text{Guess}_{\text{ser}}$ 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

$$|\Pr [\mathbf{G}_{11}^A \Rightarrow 1] - \Pr [\mathbf{G}_{12}^A \Rightarrow 1]| \leq \Pr [\text{GUESS in } \mathbf{G}_{11}^A] = \Pr [\text{GUESS in } \mathbf{G}_{12}^A]$$

Furthermore, we claim that every query to SENDRESP or SENDTERINIT will add at most one password into \mathcal{L}_{pw} . That is, at most one password will be recorded in \mathcal{L}_{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 $(\text{pw}, c, e_2, \text{enc}) \in \mathcal{L}_2$ and $(\text{pw}', c', e_2, \text{enc}) \in \mathcal{L}_2$ for some c and c' . This means that e_2 is generated by querying $E_2(\text{pw}, c)$ and $E_2(\text{pw}', c')$, which is impossible since we simulate E_2 in a 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}_{pw} .

Now we can bound the happening probability of GUESS in \mathbf{G}_{12} . A clean description of \mathbf{G}_{12} is given in Figure 21. 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

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

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

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

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

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

5 Instantiations of the Underlying KEM

5.1 Direct Diffie-Hellman-based Constructions

DIFFIE-HELLMAN ASSUMPTIONS. 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 5.1 (Multi-Instance stDH [ABR01]). 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}

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

GAME $\text{stDH}_{\mathbb{G}}^{(N,\mu),\mathcal{A}}$	Oracle $\text{PCO}(i, Y, Z)$
01 par := $(\mathbb{G}, g, p) \leftarrow \mathcal{G}$	08 if $\mathbf{X}[i] = \perp$
02 for $i \in [N]$	09 return \perp
03 $x_i \xleftarrow{\$} \mathbb{Z}_p, \mathbf{X}[i] := X_i := g^{x_i}$	10 return $Z == Y^{x_i}$
04 for $j \in [\mu]$:	
05 $y_j \xleftarrow{\$} \mathbb{Z}_p, \mathbf{Y}[j] := Y_j := g^{y_j}$	
06 $(i^*, j^*, Z^*) \leftarrow \mathcal{A}^{\text{stDH}}(\text{par}, \mathbf{X}, \mathbf{Y})$	
07 return $Z^* = Y_{j^*}^{x_{i^*}}$	

Figure 15: Security games OW-PCA and OW-rPCA for KEM scheme KEM.

is negligible, where $\text{stDH}_{\mathbb{G}}^{(N,\mu),\mathcal{A}}$ is defined in Figure 15.

CONSTRUCTION BASED ON STRONG DH. In Figure 16, we construct a KEM scheme KEM_{stDH} with plaintext space \mathbb{G} and ciphertext space of \mathbb{G} . KEM_{stDH} is essentially the hashed ElGamal KEM [ABR01, CKS08].

KG₁	Encaps₁(pk)	Decaps₁(sk, R)
01 $x \xleftarrow{\$} \mathbb{Z}_p$	06 $r \xleftarrow{\$} \mathbb{Z}_p$	11 parse $(x, \text{pk}) =: \text{sk}$
02 $X := g^x$	07 $R := g^r \in \mathbb{G}$	12 parse $R =: \text{c}$
03 $\text{pk} := X$	08 $\text{k} := \text{H}(\text{pk}, R, X^r)$	13 $\text{k} := \text{H}(\text{pk}, R, R^x)$
04 $\text{sk} := (x, \text{pk})$	09 $\text{c} := R$	14 return k
05 return (pk, sk)	10 return (c, k)	

Figure 16: KEM scheme $\text{KEM}_{\text{stDH}} = (\text{Setup}_1, \text{KG}_1, \text{Encaps}_1, \text{Decaps}_1)$.

KEM_{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

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

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 [Bha20]. 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 (N,μ) -stDH assumption.

5.2 Generic Constructions

Let $\text{PKE}_0 = (\text{KG}_0, \text{Enc}_0, \text{Dec}_0)$ 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 $\text{G} : \mathcal{PK} \times \mathcal{M} \rightarrow \mathcal{R}$, $\text{H} : \mathcal{PK} \times \mathcal{M} \times \mathcal{C} \rightarrow \{0,1\}^L$, and $\text{H}' : \mathcal{PK} \times \{0,1\}^\ell \times \mathcal{C} \rightarrow \{0,1\}^L$ be hash functions. Let $\text{PKE}_0 = (\text{Setup}_0, \text{KG}_0, \text{Enc}_0, \text{Dec}_0)$ be a PKE scheme. In Figure 17, we define a generic transformation for KEM schemes. We denote such transformation as $\text{KEM} = \text{TU}^\ell[\text{PKE}_0, \text{G}, \text{H}, \text{H}']$. TU^ℓ is essentially a combination of the T transformation and the U^ℓ transformation in [HHK17]. 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 .

CORRECTNESS OF KEM. We follow the correctness proof of [HHK17, Theorem 3.1]. Decaps has decapsulation error if its input is $\text{c} = \text{Enc}_0(\text{pk}, m'; \text{G}(\text{pk}, m'))$ for some m' and $\text{Dec}_0(\text{sk}, \text{c}) \neq m'$. If PKE_0 is $(1 - \delta_{\text{PKE}_0})$ -correct, such event happens within probability $q_{\text{G}} \cdot \delta_{\text{PKE}_0}$ if we treat G as a random oracle and assume G will be queried at most q_{G} times. Therefore, KEM is $(1 - q_{\text{G}} \cdot \delta_{\text{PKE}_0})$ -correct.

SECURITY. In Theorems 5.2 to 5.4, 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 TU^ℓ transformation preserves the public key fuzzyness of the underlying PKE.

KG(par)	Encaps(pk)	Decaps((pk, sk, s), c)
01 $(pk, sk) \leftarrow KG_0(par)$	05 $m \xleftarrow{\$} \mathcal{M}'$	10 $m' := Dec_0(sk, c)$
02 $s \xleftarrow{\$} \{0, 1\}^\ell$	06 $r := G(pk, m)$	11 if $m' \neq \perp$
03 $sk' := (pk, sk, s)$	07 $c := Enc_0(pk, m; r)$	12 and $c = Enc_0(pk, m'; G(pk, m'))$
04 return (pk, sk')	08 $k := H(pk, c, m)$	13 $k := H(pk, c, m')$
	09 return (c, k)	14 else $k := H'(pk, c, s)$
		15 return k

Figure 17: KEM scheme $KEM = (Setup, KG, Encaps, Decaps)$ from the generic transformation $TU^\chi[PKE_0, G, H, H']$, where G, H , and H' are hash functions, $PKE_0 = (Setup_0, KG_0, Enc_0, Dec_0)$ is a PKE scheme, and $Setup = Setup_0$.

Theorem 5.2 *Let N be the number of users. If PKE_0 has fuzzy public keys, then $KEM = TU^\chi[PKE_0, G, H, H']$ in Figure 17 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*

$$\text{Adv}_{KEM}^{N\text{-FUZZY}}(\mathcal{A}) \leq \text{Adv}_{PKE_0}^{N\text{-FUZZY}}(\mathcal{B})$$

Theorems 5.3 and 5.4 show that if PKE_0 is PR-CPA secure, then $KEM = TU^\chi[PKE_0, G, H, H']$ has OW-CPA security and ciphertext anonymity under PCA attacks. For readability, we postpone their proofs to Supp. Mat. B.

Theorem 5.3 *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 $KEM = TU^\chi[PKE_0, G, H, H']$ has anonymous ciphertext under PCA attacks (cf. Definition 2.7).*

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

$$\begin{aligned} \text{Adv}_{KEM}^{(N, \mu)\text{-ANO}}(\mathcal{A}) &\leq 2\text{Adv}_{PKE_0}^{(N, \mu)\text{-PR-CPA}}(\mathcal{B}) + 2Nq_G \cdot \delta + \frac{N\mu q_G}{|\mathcal{M}|} \\ &\quad + \frac{2N(q_{H'} + q_{PCO})}{2^\ell} + \frac{N^2\mu^2 + q_G^2}{|\mathcal{R}|} + \frac{2N^2\mu^2 + q_H^2 + q_{H'}^2}{2^L}, \end{aligned}$$

where $q_G, q_H, q_{H'}$, and q_{PCO} are the numbers of \mathcal{A} 's queries to G, H, H' , and PCO .

Theorem 5.4 *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 $KEM = TU^\chi[PKE_0, G, H, H']$ 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

$$\begin{aligned} \text{Adv}_{KEM}^{(N, \mu)\text{-OW-PCA}}(\mathcal{A}) &\leq 2\text{Adv}_{PKE_0}^{(N, \mu)\text{-PR-CPA}}(\mathcal{B}) + 2Nq_G \cdot \delta + \frac{N\mu(q_G + q_H)}{|\mathcal{M}|} \\ &\quad + \frac{2N(q_{H'} + q_{PCO})}{2^\ell} + \frac{N^2\mu^2 + q_G^2}{|\mathcal{R}|} + \frac{2N^2\mu^2 + q_H^2 + q_{H'}^2}{2^L}, \end{aligned}$$

where $q_G, q_H, q_{H'}$, and q_{PCO} are the numbers of \mathcal{A} 's queries to G, H, H' , and PCO .

By combining Lemma 2.8 and Theorems 5.3 and 5.4, we have Theorem 5.5.

Theorem 5.5 *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 $KEM = TU^\chi[PKE_0, G, H, H']$ 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

$$\begin{aligned} \text{Adv}_{KEM}^{(N, \mu)\text{-OW-rPCA}}(\mathcal{A}) &\leq 4\text{Adv}_{PKE_0}^{(N, \mu)\text{-PR-CPA}}(\mathcal{B}) + 4Nq_G \cdot \delta + \frac{N\mu(2q_G + q_H)}{|\mathcal{M}|} \\ &\quad + \frac{4N(q_{H'} + q_{PCO})}{2^\ell} + \frac{2(N^2\mu^2 + q_G^2)}{|\mathcal{R}|} + \frac{2(2N^2\mu^2 + q_H^2 + q_{H'}^2)}{2^L}, \end{aligned}$$

where $q_G, q_H, q_{H'}$, and q_{PCO} are the numbers of \mathcal{A} 's queries to G, H, H' , and PCO .

5.3 Lattice-based Instantiations

We discuss two lattice-based instantiations of the PAKE protocol Π (Figure 7). The first one is the well-known Regev’s encryption [Reg05, PVW08] which is based on learning with error (LWE) assumption. The second one is the Kyber.PKE scheme [SAB⁺20], 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 [Reg05, PVW08, GPV08, SAB⁺20].

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 (IC_1, IC_2) in Figure 7. Let ϵ_{LWE} and ϵ_{mlwe} be the best computational advantage against the LWE and MLWE assumptions, respectively. We use $\text{negl}(\lambda)$ to denote negligible (about λ) statistical terms. Such terms do not influence tightness.

REGEV ENCRYPTION. We use the multi-bit version of Regev’s encryption, denoted as $\text{PKE}_{\text{Regev}}$, in [PVW08]. As shown in [PVW08, 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

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

We can use the TU^\times transformation to transform $\text{PKE}_{\text{Regev}}$ into a KEM scheme and then use the KEM scheme to instantiate Π (Figure 7). By plugging these bounds into Theorems 5.3 to 5.5 and then Theorem 4.1, we have

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

KYBER PKE. We consider the Kyber.CPAPKE scheme (denoted as $\text{PKE}_{\text{kyber}}$) in [SAB⁺20]. The pseudorandomness and fuzzyness proofs of $\text{PKE}_{\text{kyber}}$ are the same as in [BCP⁺23, 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

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

By using the TU^\times transformation, we can transform $\text{PKE}_{\text{kyber}}$ into a KEM scheme. Then we use the KEM scheme to instantiate Π (Figure 7). By Theorems 4.1 and 5.3 to 5.5, we have

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

References

- [AB19] Michel Abdalla and Manuel Barbosa. Perfect forward security of SPAKE2. Cryptology ePrint Archive, Report 2019/1194, 2019. <https://eprint.iacr.org/2019/1194>. (Cited on page 9.)
- [ABB⁺20] Michel Abdalla, Manuel Barbosa, Tatiana Bradley, Stanislaw Jarecki, Jonathan Katz, and Jiayu Xu. Universally composable relaxed password authenticated key exchange. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 278–307. Springer, Heidelberg, August 2020. (Cited on page 4.)
- [ABR01] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In David Naccache, editor, *CT-RSA 2001*, volume 2020 of *LNCS*, pages 143–158. Springer, Heidelberg, April 2001. (Cited on page 4, 21, 22.)
- [AEK⁺22] Michel Abdalla, Thorsten Eisenhofer, Eike Kiltz, Sabrina Kunzweiler, and Doreen Riepel. Password-authenticated key exchange from group actions. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 699–728. Springer, Heidelberg, August 2022. (Cited on page 3, 4.)

- [AFP05] Michel Abdalla, Pierre-Alain Fouque, and David Pointcheval. Password-based authenticated key exchange in the three-party setting. In Serge Vaudenay, editor, *PKC 2005*, volume 3386 of *LNCS*, pages 65–84. Springer, Heidelberg, January 2005. (Cited on page 9.)
- [AP05] Michel Abdalla and David Pointcheval. Simple password-based encrypted key exchange protocols. In Alfred Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 191–208. Springer, Heidelberg, February 2005. (Cited on page 3.)
- [BBDQ18] Fabrice Benhamouda, Olivier Blazy, Léo Ducas, and Willy Quach. Hash proof systems over lattices revisited. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part II*, volume 10770 of *LNCS*, pages 644–674. Springer, Heidelberg, March 2018. (Cited on page 3, 4.)
- [BCP⁺23] Hugo Beguinet, Céline Chevalier, David Pointcheval, Thomas Ricosset, and Mélissa Rossi. GeT a CAKE: Generic transformations from key encapsulation mechanisms to password authenticated key exchanges. *ACNS 2023*, 2023. <https://eprint.iacr.org/2023/470>. (Cited on page 3, 4, 5, 11, 24, 34.)
- [Bha20] Rishiraj Bhattacharyya. Memory-tight reductions for practical key encapsulation mechanisms. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part I*, volume 12110 of *LNCS*, pages 249–278. Springer, Heidelberg, May 2020. (Cited on page 22.)
- [BIO⁺17] José Becerra, Vincenzo Iovino, Dimitar Ostrev, Petra Sala, and Marjan Skrobot. Tightly-secure PAK(E). In Srdjan Capkun and Sherman S. M. Chow, editors, *CANS 17*, volume 11261 of *LNCS*, pages 27–48. Springer, Heidelberg, November / December 2017. (Cited on page 3.)
- [BM92] Steven M. Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *1992 IEEE Symposium on Security and Privacy*, pages 72–84. IEEE Computer Society Press, May 1992. (Cited on page 3.)
- [BP02] Mihir Bellare and Adriana Palacio. GQ and Schnorr identification schemes: Proofs of security against impersonation under active and concurrent attacks. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 162–177. Springer, Heidelberg, August 2002. (Cited on page 3.)
- [BP18] Daniel J. Bernstein and Edoardo Persichetti. Towards KEM unification. *Cryptology ePrint Archive*, Report 2018/526, 2018. <https://eprint.iacr.org/2018/526>. (Cited on page 6.)
- [BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, Heidelberg, May 2000. (Cited on page 3, 4, 9.)
- [BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, Heidelberg, May / June 2006. (Cited on page 5.)
- [Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001. (Cited on page 34.)
- [CHK⁺05] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 404–421. Springer, Heidelberg, May 2005. (Cited on page 3.)
- [CKS08] David Cash, Eike Kiltz, and Victor Shoup. The twin Diffie-Hellman problem and applications. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 127–145. Springer, Heidelberg, April 2008. (Cited on page 4, 22.)

- [GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 197–206. ACM Press, May 2008. (Cited on page 24.)
- [HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 341–371. Springer, Heidelberg, November 2017. (Cited on page 11, 22.)
- [HL19] Björn Haase and Benoît Labrique. 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>. (Cited on page 3.)
- [HR10] Feng Hao and Peter Ryan. J-PAKE: Authenticated key exchange without PKI. Cryptology ePrint Archive, Report 2010/190, 2010. <https://eprint.iacr.org/2010/190>. (Cited on page 3.)
- [Jab96] David P. Jablon. Strong password-only authenticated key exchange. *SIGCOMM Comput. Commun. Rev.*, 26(5):526, oct 1996. (Cited on page 3.)
- [KV09] Jonathan Katz and Vinod Vaikuntanathan. Smooth projective hashing and password-based authenticated key exchange from lattices. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 636–652. Springer, Heidelberg, December 2009. (Cited on page 3.)
- [LLHG23] Xiangyu Liu, Shengli Liu, Shuai Han, and Dawu Gu. EKE meets tight security in the Universally Composable framework. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part I*, volume 13940 of *LNCS*, pages 685–713. Springer, Heidelberg, May 2023. (Cited on page 4, 5.)
- [Mac02] Philip MacKenzie. The pak suite: Protocols for password-authenticated key exchange. 12 2002. (Cited on page 3.)
- [OP01a] Tatsuaki Okamoto and David Pointcheval. The gap-problems: A new class of problems for the security of cryptographic schemes. In Kwangjo Kim, editor, *PKC 2001*, volume 1992 of *LNCS*, pages 104–118. Springer, Heidelberg, February 2001. (Cited on page 3.)
- [OP01b] Tatsuaki Okamoto and David Pointcheval. REACT: Rapid Enhanced-security Asymmetric Cryptosystem Transform. In David Naccache, editor, *CT-RSA 2001*, volume 2020 of *LNCS*, pages 159–175. Springer, Heidelberg, April 2001. (Cited on page 4, 6.)
- [PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, Heidelberg, August 2008. (Cited on page 24.)
- [Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005. (Cited on page 4, 24.)
- [SAB⁺20] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, and Damien Stehlé. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2020. available at <https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions>. (Cited on page 4, 11, 24.)
- [SGJ23] Bruno Freitas Dos Santos, Yanqi Gu, and Stanislaw Jarecki. Randomized half-ideal cipher on groups with applications to UC (a)PAKE. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 128–156. Springer, Heidelberg, April 2023. (Cited on page 5, 11.)

- [ZY17] Jiang Zhang and Yu Yu. Two-round PAKE from approximate SPH and instantiations from lattices. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 37–67. Springer, Heidelberg, December 2017. (Cited on page 3.)

Supporting Material

A Omitted Proofs in Section 4.1

Lemma 4.4. To upper bound $|\Pr[\mathbf{G}_1^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathbf{G}_2^{\mathcal{A}} \Rightarrow 1]|$, we construct a direct reduction \mathcal{B}_1 from the fuzzyness of KEM in Figure 18. \mathcal{B}_1 uses the input \mathbf{pk} to simulate the outputs of \mathbf{D}_1 (cf. Line 11). Since \mathcal{A} 's number of queries to \mathbf{D}_1 is q_1 , we need at most q_1 public keys from the fuzzyness game.

Reduction $\mathcal{B}_1(\text{par}, \mathbf{pk})$	Oracle $\mathbf{D}_1(\text{pw}, e_1)$
01 $\text{cnt} := 0$	08 if $\exists(\text{pw}, \mathbf{pk}, e_1, *) \in \mathcal{L}_1$
02 for $(\text{U}, \text{S}) \in \mathcal{U} \times \mathcal{S}$	09 return \mathbf{pk}
03 $\text{pw}_{\text{U}, \text{S}} \leftarrow \mathcal{PW}$	10 $\text{cnt} := \text{cnt} + 1$
04 $\mathcal{C} := \emptyset$	11 $\mathbf{pk} := \mathbf{pk}[\text{cnt}]$
05 $\beta \leftarrow \{0, 1\}$	12 $\mathcal{L}_{\text{key}} := \mathcal{L}_{\text{key}} \cup \{(\mathbf{pk}, \perp)\}$
06 $b' \leftarrow \mathcal{A}^{\text{O}, \text{H}, \text{IC}_1, \text{IC}_2}(\text{par})$	13 $\mathcal{L}_1 := \mathcal{L}_1 \cup (\text{pw}, \mathbf{pk}, e_1, \text{dec})$
07 return $\beta == b'$	14 return \mathbf{pk}

Figure 18: Reduction \mathcal{B}_1 in proving Lemma 4.4. Highlighted parts show how we use the challenge input to simulate \mathbf{G}_1 or \mathbf{G}_2 for \mathcal{A} . All other oracles, except \mathbf{D}_1 , are the same as shown in Figure 8.

If \mathbf{pk} is generated from uniformly sampling from \mathcal{PK} , then \mathcal{B}_1 simulates \mathbf{G}_1 . If \mathbf{pk} is generated from KG, then \mathcal{B}_1 simulates \mathbf{G}_2 . \mathcal{B}_1 does not need secret keys of \mathbf{pk} to simulate these games. Therefore, we have

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

□

Lemma 4.5. We construct a direct reduction \mathcal{B}_3 to upper bound the probability difference between \mathbf{G}_3 and \mathbf{G}_4 . \mathcal{B}_3 plays ANO-PCA $_{\text{KEM}, b}^{(S, 1)}$ (cf. Figure 19).

Reduction $\mathcal{B}_3(\text{par}, \mathbf{pk}, \mathbf{c})$	Oracle EXECUTE($\text{U}, t_1, \text{S}, t_2$)
01 $\text{cnt} := 0$	08 if $\pi_{\text{U}}^{t_1} \neq \perp$ or $\pi_{\text{S}}^{t_2} \neq \perp$: return \perp
02 for $(\text{U}, \text{S}) \in \mathcal{U} \times \mathcal{S}$	09 $\text{cnt} := \text{cnt} + 1$, $\text{pw} := \text{pw}_{\text{U}, \text{S}}$
03 $\text{pw}_{\text{U}, \text{S}} \leftarrow \mathcal{PW}$	10 $\mathbf{pk} := \mathbf{pk}[\text{cnt}]$, $e_1 := \text{E}_1(\text{pw}, \mathbf{pk})$
04 $\mathcal{C} := \emptyset$	11 $\mathbf{c} := \mathbf{c}[\text{cnt}, 1]$, $e_2 := \text{E}_2(\text{pw}, \mathbf{c})$
05 $\beta \leftarrow \{0, 1\}$	12 $\text{ctxt} := (\text{U}, \text{S}, e_1, e_2)$, $\text{SK} \stackrel{\$}{\leftarrow} \text{SK}$
06 $b' \leftarrow \mathcal{A}^{\text{O}, \text{H}, \text{IC}_1, \text{IC}_2}(\text{par})$	13 $\pi_{\text{U}}^{t_1} := ((\perp, \perp, e_1), \text{ctxt}, \text{SK}, \text{true})$, $\pi_{\text{U}}^{t_1}.\text{fr} := \text{true}$
07 return $\beta == b'$	14 $\pi_{\text{S}}^{t_2} := ((\perp, \perp, e_2), \text{ctxt}, \text{SK}, \text{true})$, $\pi_{\text{S}}^{t_2}.\text{fr} := \text{true}$
	15 return $(\text{U}, e_1, \text{S}, e_2)$

Figure 19: Reduction \mathcal{B}_3 in proving Lemma 4.5. Highlighted parts show how we use the challenge input to simulate \mathbf{G}_3 or \mathbf{G}_4 for \mathcal{A} . All other oracles, except EXECUTE, are the same as shown in \mathbf{G}_3 (cf. Figure 8 and Figure 9).

\mathcal{B}_3 uses the inputs \mathbf{pk} and \mathbf{c} to simulate EXECUTE (cf. Line 11). Since \mathcal{A} 's issues S queries to EXECUTE, we need at most S public keys from the ciphertext anonymity security game. If \mathcal{B}_3 plays ANO-PCA $_{\text{KEM}, 0}^{(S, 1)}$, then \mathbf{c} is generated by uniform sampling from \mathcal{C} , which means that \mathcal{B}_3 is simulating \mathbf{G}_3 . Otherwise, \mathbf{c} is generated from encapsulating \mathbf{pk} and \mathcal{B}_3 simulates \mathbf{G}_4 . \mathcal{B}_3 does not need secret keys of \mathbf{pk} and decapsulated keys of \mathbf{c} (and thus does not need to PCO oracle) to simulate these games. Therefore, we have

$$|\Pr[\mathbf{G}_3^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathbf{G}_4^{\mathcal{A}} \Rightarrow 1]| \leq \text{Adv}_{\text{KEM}}^{((S, 1)\text{-ANO}}(\mathcal{B}_3) + \frac{S^2}{|\mathcal{C}|}$$

□

<p>Game $\mathcal{B}_6(\text{par}, \text{pk}, \text{c})$</p> <pre> 01 for (U, S) ∈ U: pw_{U,S} ← PW 02 C := ∅, β ← {0, 1} 03 Guess_{user} := false, Guess_{ser} := false 04 b' ← A^{O,H,IC₁,IC₂}(par) 05 return β == b'</pre> <p>Oracle SENDINIT(U, t₁, S, e₂)</p> <pre> 06 if π_U^{t₁} = ⊥ and π_U^{t₁}.tr ≠ (U, S, *, *) 07 return ⊥ 08 (pk, i, e₁) := π_U^{t₁}.e, c := D₂(pw_{U,S}, e₂) 09 if ∃ t₂ s.t. π_S^{t₂}.fr = true 10 and π_S^{t₂}.tr = (U, S, e₁, e₂) 11 π_U^{t₁}.fr := true, SK := π_S^{t₂}.key 12 else 13 ctxt := (U, S, e₁, e₂) 14 if (U, S) ∉ C 15 π_U^{t₁}.fr := true 16 if e₂ ∉ L₂^S and ∃ c s.t. (pw_{U,S}, c, e₂, enc) ∈ L₂ 17 Guess_{user} := true 18 SK := Patch(ctxt, pk, i, c) 19 else SK $\stackrel{\\$}{\leftarrow}$ SK 20 else 21 π_U^{t₁}.fr := false 22 SK := Patch(ctxt, pk, i, c) 23 π_U^{t₁}.tr, key, acc := (ctxt, SK, true) 24 return true</pre>	<p>Oracle D₁(pw, e₁)</p> <pre> 25 if ∃ (pw, pk, e₁, *) ∈ L₁ 26 return c 27 cnt₂[cnt₁] := 0, cnt₁ := cnt₁ + 1 28 pk := pk[cnt₁], L_{key} := L_{key} ∪ {(pk, cnt₁)}</pre> <p>Oracle SENDRESP(S, t₂, U, e₁)</p> <pre> 31 π_S^{t₂} ≠ ⊥: return ⊥ 32 pk := D₁(pw_{U,S}, e₁) 33 if (U, S) ∈ C 34 π_S^{t₂}.fr := false 35 (c, k) ← Encaps(pk), e₂ := E₂(pw_{U,S}, c) 36 ctxt := (U, S, e₁, e₂) 37 SK := H(ctxt, pk, c, k, pw_{U,S}) 38 else 39 π_S^{t₂}.fr := true 40 if e₁ ∉ L₁^U and ∃ pk s.t. (pw_{U,S}, pk, e₁, enc) ∈ L₁ 41 Guess_{ser} := true 42 (c, k) ← Encaps(pk), e₂ := E₂(pw_{U,S}, c) 43 SK := H(ctxt, pk, c, k, pw_{U,S}) 44 else 45 Retrieve i s.t. (pk, i) ∈ L_{key} 46 cnt₂[i] := cnt₂[i] + 1, j := cnt₂[i] 47 c := c[i, j], e₂ := E₂(pw_{U,S}, c) 48 SK $\stackrel{\\$}{\leftarrow}$ SK 49 L₂^S := L₂^S ∪ {e₂} 50 π_S^{t₂} := ((c, k, e₂), ctxt, SK, true) 51 return (S, e₂)</pre>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Figure 20: Reduction \mathcal{B}_6 in bounding the probability difference between \mathbf{G}_9 and \mathbf{G}_{10} . Highlighted parts show how \mathcal{B}_6 uses PCO and challenge input to simulate \mathbf{G}_{10} . All other oracles not shown in the figure are the same as in \mathbf{G}_8 (cf. Figures 8, 9 and 11). Procedure Patch is the same as the one shown in Figure 12.

Lemma 4.6. We construct a reduction \mathcal{B}_6 to upper bound the probability difference between \mathbf{G}_9 and \mathbf{G}_{10} . \mathcal{B}_6 is less straight-forward than \mathcal{B}_3 since here \mathcal{B}_6 needs to use the PCO oracle to simulate the session keys of user instances (as we did in constructing \mathcal{B}_5 , cf. Figure 13).

On input $(\text{par}, \text{pk}, \text{c})$, \mathcal{B}_6 embeds challenge public keys pk into queries to D_1 (cf. Line 28). By Lines 34 to 36, public keys generated in SENDINIT are also from pk . Similar to \mathcal{B}_4 and \mathcal{B}_5 , \mathcal{B}_6 uses the Patch procedure in Figure 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.

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 from the ciphertext anonymity security game. The challenge ciphertexts c are embedded in Line 47. If \mathcal{B}_6 plays ANO-PCA_{KEM,0}^(S+q₁,S), then c is generated by uniform sampling from \mathcal{C} , which means that \mathcal{B}_6 is simulating \mathbf{G}_{10} . Otherwise, c is generated from encapsulating pk and \mathcal{B}_6 simulates \mathbf{G}_9 . We have

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

□

<p>Game G_{12}</p> <p>01 par \leftarrow Setup</p> <p>02 $\mathcal{C} := \emptyset, \beta \leftarrow \{0, 1\}$</p> <p>03 $\text{Guess}_{\text{user}} := \text{false}, \text{Guess}_{\text{ser}} := \text{false}$</p> <p>04 $b' \leftarrow \mathcal{A}^{O, H, IC_1, IC_2}(\text{par})$</p> <p>05 for $(U, S) \in \mathcal{U} \times \mathcal{S}$</p> <p>06 if $(U, S) \notin \mathcal{C}$: $\text{pw}_{U,S} \leftarrow \mathcal{PW}$</p> <p>07 if $\exists S'$ s.t. $\text{pw}_{U,S'} \in \mathcal{L}_{\text{pw}}$</p> <p>08 $\text{Guess}_{\text{user}} := \text{true}$</p> <p>09 if $\exists U'$ s.t. $\text{pw}_{U',S} \in \mathcal{L}_{\text{pw}}$</p> <p>10 $\text{Guess}_{\text{ser}} := \text{true}$</p> <p>11 return $\beta == b'$</p> <p>Oracle REVEAL(P, t)</p> <p>12 if $\pi_P^t.\text{acc} \neq \text{true}$ or $\pi_P^t.\text{test} = \text{true}$</p> <p>13 return \perp</p> <p>14 if $\exists P'$ $\in \mathcal{U} \cup \mathcal{S}, t'$ s.t.</p> <p>15 Partner($\pi_P^t, \pi_{P'}^{t'}$) = true</p> <p>16 and $\pi_{P'}^{t'}.\text{test} = \text{true}$</p> <p>17 return \perp</p> <p>18 for $\forall (P', t')$ s.t. $\pi_{P'}^{t'}.\text{tr} = \pi_P^t.\text{tr}$</p> <p>19 $\pi_{P'}^{t'}.\text{fr} := \text{false}$</p> <p>20 return $\pi_P^t.\text{key}$</p> <p>Oracle TEST(P, t)</p> <p>21 if $\pi_P^t.\text{fr} = \text{false}$: return \perp</p> <p>22 $\text{SK}_0^* := \text{REVEAL}(P, t), \text{SK}_1^* \leftarrow^{\\$} \text{SK}$</p> <p>23 if $\text{SK}_0^* = \perp$: return \perp</p> <p>24 $\pi_P^t.\text{test} := \text{true}$</p> <p>25 return SK_β^*</p> <p>Oracle CORRUPT(U, S)</p> <p>26 if $(U, S) \in \mathcal{C}$: return \perp</p> <p>27 $\mathcal{C} := \mathcal{C} \cup \{(U, S)\}$</p> <p>28 $\text{pw}_{U,S} \leftarrow \mathcal{PW}$</p> <p>29 return $\text{pw}_{U,S}$</p> <p>Oracle E_1(pw, pk)</p> <p>30 if $\exists (\text{pw}, \text{pk}, e_1, *) \in \mathcal{L}_1$: return e_1</p> <p>31 $e_1 \leftarrow^{\\$} \mathcal{E}_1 \setminus \mathcal{T}_1, \mathcal{T}_1 := \mathcal{T}_1 \cup \{e_1\}$</p> <p>32 $\mathcal{L}_1 := \mathcal{L}_1 \cup (\text{pw}, \text{pk}, e_1, \text{enc})$</p> <p>33 return e_1</p> <p>Oracle E_2(pw, c)</p> <p>34 if $\exists (\text{pw}, c, e_2, *) \in \mathcal{L}_2$: return e_2</p> <p>35 $e_2 \leftarrow^{\\$} \mathcal{E}_2 \setminus \mathcal{T}_2, \mathcal{T}_2 := \mathcal{T}_2 \cup \{e_2\}$</p> <p>36 $\mathcal{L}_2 := \mathcal{L}_2 \cup (\text{pw}, c, e_2, \text{enc})$</p> <p>37 return e_2</p> <p>Oracle D_1(pw, e_1)</p> <p>38 if $\exists (\text{pw}, \text{pk}, e_1, *) \in \mathcal{L}_1$</p> <p>39 return pk</p> <p>40 $(\text{pk}, \text{sk}) \leftarrow \text{KG}$</p> <p>41 $\mathcal{L}_{\text{key}} := \mathcal{L}_{\text{key}} \cup \{(\text{pk}, \text{sk})\}$</p> <p>42 $\mathcal{L}_1 := \mathcal{L}_1 \cup \{(\text{pw}, \text{pk}, e_1, \text{dec})\}$</p> <p>43 return pk</p> <p>Oracle D_2(pw, e_2)</p> <p>44 if $\exists (\text{pw}, c, e_2, *) \in \mathcal{L}_2$: return c</p> <p>45 $c \leftarrow^{\\$} \mathcal{C}, \mathcal{L}_2 := \mathcal{L}_2 \cup (\text{pw}, c, e_2, \text{dec})$</p> <p>46 return c</p>	<p>Oracle EXECUTE(U, t_1, S, t_2)</p> <p>47 if $\pi_U^{t_1} \neq \perp$ or $\pi_S^{t_2} \neq \perp$</p> <p>48 return \perp</p> <p>49 $e_1 \leftarrow^{\\$} \mathcal{E}_1 \setminus \mathcal{T}_1, \mathcal{T}_1 := \mathcal{T}_1 \cup \{e_1\}$</p> <p>50 $e_2 \leftarrow^{\\$} \mathcal{E}_2 \setminus \mathcal{T}_2, \mathcal{T}_2 := \mathcal{T}_2 \cup \{e_2\}$</p> <p>51 $\text{ctxt} := (U, S, e_1, e_2), \text{SK} \leftarrow^{\\$} \text{SK}$</p> <p>52 $\pi_U^{t_1} := ((\perp, \perp, e_1), \text{ctxt}, \text{SK}, \text{true})$</p> <p>53 $\pi_S^{t_2} := ((\perp, \perp, e_2), \text{ctxt}, \text{SK}, \text{true})$</p> <p>54 $(\pi_U^{t_1}.\text{fr}, \pi_S^{t_2}.\text{fr}) := (\text{true}, \text{true})$</p> <p>55 return (U, e_1, S, e_2)</p> <p>Oracle SENDINIT(U, t_1, S)</p> <p>56 if $\pi_U^{t_1} \neq \perp$: return \perp</p> <p>57 $e_1 \leftarrow^{\\$} \mathcal{E}_1 \setminus \mathcal{T}_1, \mathcal{T}_1 := \mathcal{T}_1 \cup \{e_1\}, \mathcal{L}_1^U := \mathcal{L}_1^U \cup \{e_1\}$</p> <p>58 $\pi_U^{t_1} := ((\perp, \perp, e_1), (U, S, e_1, \perp), \perp, \perp)$</p> <p>59 $\pi_U^{t_1}.\text{fr} := \text{false}$</p> <p>60 return (U, e_1)</p> <p>Oracle SENDRESP(S, t_2, U, e_1)</p> <p>61 $\pi_S^{t_2} \neq \perp$: return \perp</p> <p>62 if $(U, S) \in \mathcal{C}$</p> <p>63 $\pi_S^{t_2}.\text{fr} := \text{false}$</p> <p>64 $\text{pk} := D_1(\text{pw}_{U,S}, e_1), (c, k) \leftarrow \text{Encaps}(\text{pk})$</p> <p>65 $e_2 := E_2(\text{pw}_{U,S}, c), \text{ctxt} := (U, S, e_1, e_2)$</p> <p>66 $\text{SK} := H(\text{ctxt}, \text{pk}, c, k, \text{pw}_{U,S})$</p> <p>67 else</p> <p>68 $\pi_S^{t_2}.\text{fr} := \text{true}, \text{SK} \leftarrow^{\\$} \text{SK}$</p> <p>69 if $e_1 \notin \mathcal{L}_1^U$</p> <p>70 for (pw, pk) s.t. $(\text{pw}, \text{pk}, e_1, \text{enc}) \in \mathcal{L}_1$</p> <p>71 $\mathcal{L}_{\text{pw}} := \mathcal{L}_{\text{pw}} \cup \{\text{pw}\}$</p> <p>72 $e_2 \leftarrow^{\\$} \mathcal{E}_2 \setminus \mathcal{T}_2, \mathcal{T}_2 := \mathcal{T}_2 \cup \{e_2\}$</p> <p>73 $\pi_S^{t_2} := ((\perp, \perp, e_2), \text{ctxt}, \text{SK}, \text{true})$</p> <p>74 $\mathcal{L}_2^S := \mathcal{L}_2^S \cup \{e_2\}$</p> <p>75 return (S, e_2)</p> <p>Oracle SENDTERINIT(U, t_1, S, e_2)</p> <p>76 if $\pi_U^{t_1} = \perp$ and $\pi_U^{t_1}.\text{tr} \neq (U, S, *, *)$</p> <p>77 return \perp</p> <p>78 if $\exists t_2$ s.t. $\pi_S^{t_2}.\text{fr} = \text{true}$</p> <p>79 and $\pi_S^{t_2}.\text{tr} = (U, S, e_1, e_2)$</p> <p>80 $\pi_U^{t_1}.\text{fr} := \text{true}, \text{SK} := \pi_S^{t_2}.\text{key}$</p> <p>81 else</p> <p>82 $\text{ctxt} := (U, S, e_1, e_2)$</p> <p>83 if $(U, S) \notin \mathcal{C}$</p> <p>84 $\pi_U^{t_1}.\text{fr} := \text{true}, \text{SK} \leftarrow^{\\$} \text{SK}$</p> <p>85 if $e_2 \notin \mathcal{L}_2^S$</p> <p>86 for (pw, c) s.t. $(\text{pw}, c, e_2, \text{enc}) \in \mathcal{L}_2$</p> <p>87 $\mathcal{L}_{\text{pw}} := \mathcal{L}_{\text{pw}} \cup \{\text{pw}\}$</p> <p>88 else</p> <p>89 $\pi_U^{t_1}.\text{fr} := \text{false}$</p> <p>90 $\text{pk} := D_1(\text{pw}_{U,S}, e_1)$</p> <p>91 Retrieve sk s.t. $(\text{pk}, \text{sk}) \in \mathcal{L}_{\text{key}}$</p> <p>92 $c := D_2(\text{pw}_{U,S}, e_2), k := \text{Decaps}(\text{sk}, c)$</p> <p>93 $\text{SK} := H(\text{ctxt}, \text{pk}, c, k, \text{pw}_{U,S})$</p> <p>94 $\pi_U^{t_1}.\text{tr} := (\text{tr}, \text{key}, \text{acc}) := (\text{ctxt}, \text{SK}, \text{true})$</p> <p>95 return true</p> <p>Oracle H($U, S, e_1, e_2, \text{pk}, c, k, \text{pw}$)</p> <p>96 if $\mathcal{L}_H[U, S, e_1, e_2, \text{pk}, c, k, \text{pw}] = \perp$</p> <p>97 $\mathcal{L}_H[U, S, e_1, e_2, \text{pk}, c, k, \text{pw}] := \text{SK} \leftarrow^{\\$} \text{SK}$</p> <p>98 return $\mathcal{L}_H[U, S, e_1, e_2, \text{pk}, c, k, \text{pw}]$</p>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Figure 21: Game G_{12} in proving Theorem 4.1.

Games \mathbf{G}_0 - \mathbf{G}_7		PCO(i, c, k)	
01 $\text{par} \leftarrow \text{Setup}$		20 $\text{pk} := \text{pk}[i]$	
02 $\text{Query} := \text{false}$	\mathbf{G}_4 - \mathbf{G}_x	21 $m' := \text{Dec}_0(\text{sk}, c)$	\mathbf{G}_0
03 for $i \in [N]$		22 if $m' \neq \perp$	\mathbf{G}_0
04 $(\text{pk}[i], \text{sk}[i]) := (\text{pk}, \text{sk}) \leftarrow \text{KG}(\text{par})$		23 and $c = \text{Enc}_0(\text{pk}, m'; \mathbf{G}(\text{pk}, m'))$	
05 $s_i \xleftarrow{\$} \{0, 1\}^\ell$		24 $k' := \text{H}(\text{pk}, c, m')$	\mathbf{G}_0
06 for $j \in [\mu]$:		25 $k' = \text{h}^*(\text{pk}, c)$	\mathbf{G}_1 - \mathbf{G}_2
07 $m_{i,j} \xleftarrow{\$} \mathcal{M}'$		26 else	\mathbf{G}_0 - \mathbf{G}_2
08 $r_{i,j} := \mathbf{G}(\text{pk}, m_{i,j})$		27 $k' := \text{H}'(\text{pk}, c, s_i)$	\mathbf{G}_0 - \mathbf{G}_1
09 $r_{i,j} \xleftarrow{\$} \mathcal{R}$	\mathbf{G}_4 - \mathbf{G}_x	28 $k' := \text{h}^*(\text{pk}, c)$	\mathbf{G}_2
10 $c_{i,j} := \text{Enc}_0(\text{pk}, m_{i,j}; r_{i,j})$		29 $k' := \text{h}^*(\text{pk}, c)$	\mathbf{G}_3 - \mathbf{G}_x
11 $c_{i,j} \xleftarrow{\$} \mathcal{C}$	\mathbf{G}_5 - \mathbf{G}_x	30 return $k = k'$	
12 $k := \text{H}(\text{pk}_{i,j}, c_{i,j}, m_{i,j})$		$\text{H}'(\text{pk}, c, s)$	
13 $c[i, j] := c$		31 return $\text{h}'(\text{pk}, c, s)$	
14 $b' \leftarrow \mathcal{A}^{\text{PCO}, \mathbf{G}, \text{H}, \text{H}' }(\text{par}, \text{pk}, c)$		$\text{H}(\text{pk}, c, m)$	
15 return b'		32 if $c = \text{PKE}_0(\text{pk}, m; \mathbf{G}(\text{pk}, m))$	\mathbf{G}_1 - \mathbf{G}_x
$\mathbf{G}(\text{pk}, m)$		33 return $\text{h}^*(\text{pk}, c)$	\mathbf{G}_1 - \mathbf{G}_x
16 if $\exists(i, j)$ s.t. $m = m_{i,j}$	\mathbf{G}_4 - \mathbf{G}_x	34 return $\text{h}(\text{pk}, c, m)$	
17 $\text{Query} := \text{true}$	\mathbf{G}_4 - \mathbf{G}_x		
18 abort	\mathbf{G}_4 - \mathbf{G}_x		
19 return $\text{g}(\text{pk}, m)$			

Figure 22: Games \mathbf{G}_0 - \mathbf{G}_6 for proving Theorem 5.3.

B Omitted Proofs in Section 5.2

Theorem 5.3. Theorem 5.3 is proved by the game sequences in Figure 22. Slightly different from the previous proofs, in this proof, we do not use lazy sampling to simulate random oracles $\mathbf{G}, \text{H}, \text{H}'$. Instead, we assume that the game simulator has access to internal random oracles g, h , and h' and it uses these internal oracles to simulate \mathbf{G}, H , and H' , respectively. This assumption is without loss of generality, since such internal oracles can be simulated by lazy sampling.

We also assume that there is no collision among all $r_{i,j}$'s and the outputs of random oracles. This assumption adds collision bounds $\frac{N^2\mu^2 + q_G^2}{|\mathcal{R}|} + \frac{2N^2\mu^2 + q_H^2 + q_{H'}^2}{2^L}$ to the final bound of our proof. We have

$$\Pr \left[\text{ANO-PCA}_{\text{KEM},0}^{(N,\mu),\mathcal{A}} \Rightarrow 1 \right] = \Pr \left[\mathbf{G}_0^{\mathcal{A}} \Rightarrow 1 \right]$$

Game \mathbf{G}_1 . Let h^* be an internal random oracle. If \mathcal{A} queries H on (pk, c, m) such that $\text{Enc}(\text{pk}, m; \mathbf{G}(\text{pk}, m)) = c$, then the oracle returns $\text{h}^*(\text{pk}, c)$ instead of $\text{h}(\text{pk}, c, m)$ (cf. Lines 32 to 33 and Line 25).

If Enc_0 has a perfect correctness, then $\text{Enc}_0(\text{pk}, \cdot; \mathbf{G}(\text{pk}, \cdot))$ is injective and H in \mathbf{G}_1 still behaves as a random oracle. To distinguish \mathbf{G}_1 and \mathbf{G}_0 , \mathcal{A} needs to find collisions of $\text{Enc}_0(\text{pk}, \cdot; \mathbf{G}(\text{pk}, \cdot))$ for some $\text{pk} \in \text{pk}$. If \mathcal{A} finds messages (m_0, m_1) such that $\text{Enc}_0(\text{pk}, m_0; \mathbf{G}(\text{pk}, m_0)) = \text{Enc}_0(\text{pk}, m_1; \mathbf{G}(\text{pk}, m_1))$, then either m_0 or m_1 makes \mathcal{A} win the COR_{PKE} (i.e., break the correctness of PKE_0 (cf. Definition 3.1)). Since \mathcal{A} queries \mathbf{G} at most q_G times and there are N public keys during the game, we have

$$\left| \Pr \left[\mathbf{G}_0^{\mathcal{A}} \Rightarrow 1 \right] - \Pr \left[\mathbf{G}_1^{\mathcal{A}} \Rightarrow 1 \right] \right| \leq Nq_G \cdot \delta$$

Game \mathbf{G}_2 . If \mathcal{A} queries PCO on (i, c, k) where c is invalid ciphertext with respect to $\text{pk}[i]$, then we return $\text{h}^*(\text{pk}[i], c)$ instead of $\text{H}'(\text{pk}[i], c, s_i)$ (cf. Lines 27 to 28). \mathcal{A} detects this change only if it queries H' on (pk, c, s_i) for some $i \in [N]$. Since s_i is sampled at uniformly at random and the numbers of \mathcal{A} 's queries to H' and PCO are $q_{H'}$, q_{PCO} , respectively, by a union bound, we have

$$\left| \Pr \left[\mathbf{G}_1^{\mathcal{A}} \Rightarrow 1 \right] - \Pr \left[\mathbf{G}_2^{\mathcal{A}} \Rightarrow 1 \right] \right| \leq \frac{N(q_{H'} + q_{\text{PCO}})}{2^\ell}$$

Game \mathbf{G}_3 . We change PCO so that it computes $k' := \text{h}^*(\text{pk}[i], c)$ regardless of the validity of c and then compares with the query k (cf. Line 29). This change is conceptual, so we have $\Pr \left[\mathbf{G}_2^{\mathcal{A}} \Rightarrow 1 \right] =$

$\Pr [\mathbf{G}_3^{\mathcal{A}} \Rightarrow 1]$.

Game \mathbf{G}_4 . We add an abort event **Query** and change the generation of \mathbf{c} . If \mathcal{A} queries $\mathbf{G}(\mathbf{pk}, m)$ wherer m is the plaintext of one of challenge ciphertexts \mathbf{c} , then we set the flag **Query** as **true** and abort the game (cf. Lines 16 to 18). Moreover, we generate \mathbf{c} using uniform randomnesses independent of \mathbf{G} (cf. Line 09). Let Query_i be the event that **Query** is set as **true** in \mathbf{G}_i . If Query_4 does not happen, then \mathbf{G}_4 is equivalent to \mathbf{G}_5 , so we have

$$|\Pr [\mathbf{G}_3^{\mathcal{A}} \Rightarrow 1] - \Pr [\mathbf{G}_4^{\mathcal{A}} \Rightarrow 1]| \leq \Pr [\text{Query}_4]$$

Game \mathbf{G}_5 . We change the generation of challenge ciphertexts. In this game, \mathbf{c} is sampled uniformly at random from \mathcal{C} (cf. Line 11).

To upper bound the difference between \mathbf{G}_4 and \mathbf{G}_5 , we construct a direct reduction $\mathcal{B} := (\mathcal{B}_0, \mathcal{B}_1)$ in Figure 23 to attack the PR-CPA security of PKE_0 .

<u>Reduction $\mathcal{B}_0(\text{par}, \mathbf{pk})$</u>	<u>$\mathbf{G}(\mathbf{pk}, m)$</u>	<u>$\text{PCO}(i, \mathbf{c}, k)$</u>
01 Query := false	10 if $\exists(i, j)$ s.t. $m = \mathbf{m}[i, j]$	17 $\mathbf{pk} := \mathbf{pk}[i]$
02 for $(i, j) \in [N] \times [\mu]$	11 Query := true	18 $k' := \mathbf{h}^*(\mathbf{pk}, \mathbf{c})$
03 $\mathbf{m}[i, j] := m_{i,j} \stackrel{\mathcal{L}}{\leftarrow} \mathcal{M}'$	12 abort	19 return $k == k'$
04 st := $(\text{par}, \mathbf{pk}, \mathbf{m})$	13 return $\mathbf{g}(\mathbf{pk}, m)$	<u>$\mathbf{H}'(\mathbf{pk}, \mathbf{c}, s)$</u>
05 return (\mathbf{c}, st)	<u>$\mathbf{H}(\mathbf{pk}, \mathbf{c}, m)$</u>	20 return $\mathbf{h}'(\mathbf{pk}, \mathbf{c}, s)$
<u>Reduction $\mathcal{B}_1(\text{st}, \mathbf{c})$</u>	14 if $\mathbf{c} = \mathbf{H}(\mathbf{pk}, \mathbf{c}, m)$	
06 Query := false	$\text{PKE}_0(\mathbf{pk}, m; \mathbf{G}(\mathbf{pk}, m))$	
07 parse $(\text{par}, \mathbf{pk}, \mathbf{m}) := \text{st}$	15 return $\mathbf{h}^*(\mathbf{pk}, \mathbf{c})$	
08 $b' \leftarrow \mathcal{A}^O(\text{par}, \mathbf{pk}, \mathbf{c})$	16 return $\mathbf{h}(\mathbf{pk}, \mathbf{c}, m)$	
09 return b'		

Figure 23: Reduction \mathcal{B} in bounding the probability difference between \mathbf{G}_4 and \mathbf{G}_5 in the proof of Theorem 5.3. \mathcal{A} has access to $O := \{\text{PCO}, \mathbf{G}, \mathbf{H}, \mathbf{H}'\}$.

If \mathcal{B} plays $\text{PR-CPA}_{\text{PKE}_0,0}^{(N,\mu),\mathcal{A}}$, then \mathbf{c} generated by encrypting \mathbf{m} , and thus \mathcal{B} simulates \mathbf{G}_4 for \mathcal{A} . Otherwise, \mathbf{c} is uniformly random, and \mathcal{B} simulates \mathbf{G}_5 for \mathcal{A} . Therefore, we have

$$\begin{aligned} & |\Pr [\mathbf{G}_4^{\mathcal{A}} \Rightarrow 1] - \Pr [\mathbf{G}_5^{\mathcal{A}} \Rightarrow 1]| \\ &= \left| \Pr [\text{PR-CPA}_{\text{PKE}_0,0}^{(N,\mu),\mathcal{B}} \Rightarrow 1] - \Pr [\text{PR-CPA}_{\text{PKE}_0,1}^{(N,\mu),\mathcal{B}} \Rightarrow 1] \right| = \text{Adv}_{\text{PKE}_0}^{(N,\mu)\text{-PR-CPA}}(\mathcal{B}) \end{aligned}$$

Similarly, we can change \mathcal{B} that it outputs 1 if **Query** is set as **true** (which can be efficiently determined), so we also have

$$|\Pr [\text{Query}_4] - \Pr [\text{Query}_5]| = \text{Adv}_{\text{PKE}_0}^{(N,\mu)\text{-PR-CPA}}(\mathcal{B})$$

Moreover, in \mathbf{G}_5 , $m_{i,j}$'s are independent of \mathcal{A} 's view, by a union bound, we have

$$\Pr [\text{Query}_5] \leq N\mu q_{\mathbf{G}} / |\mathcal{M}|$$

Now $\mathbf{G}_5^{\mathcal{A}}$ is equivalent to $\text{ANO-PCA}_{\text{KEM},1}^{(N,\mu),\mathcal{A}}$ if we undo the changes made in $\mathbf{G}_1\text{-}\mathbf{G}_4$. By combining all probability differences, we have

$$\begin{aligned} \text{Adv}_{\text{KEM}}^{(N,\mu)\text{-ANO}}(\mathcal{A}) &\leq 2\text{Adv}_{\text{PKE}_0}^{(N,\mu)\text{-PR-CPA}}(\mathcal{B}) + 2Nq_{\mathbf{G}} \cdot \delta + \frac{N\mu q_{\mathbf{G}}}{|\mathcal{M}|} \\ &\quad + \frac{2N(q_{\mathbf{H}'} + q_{\text{PCO}})}{2^\ell} + \frac{N^2\mu^2 + q_{\mathbf{G}}^2}{|\mathcal{R}|} + \frac{2N^2\mu^2 + q_{\mathbf{H}}^2 + q_{\mathbf{H}'}^2}{2^L} \end{aligned}$$

□

Theorem 5.4. The proof of Theorem 5.3 can be adapted to OW-PCA security proof of KEM. This is because, to break the anonymity of KEM, the adversary needs to invert the challenge ciphertext and find the plaintext (which breaks the PR-CPA security of PKE₀). Similarly, if we model \mathbf{H} as a random oracle, then to break the OW-PCA security of KEM, the adversary also needs to invert the challenge ciphertext. Concretely, in \mathbf{G}_4 (Figure 22), we can add similar abort event in oracle \mathbf{H} . This only change the happening probability of Query_5 , which will become $\Pr[\text{Query}_5] \leq N\mu(q_G + q_H)/|\mathcal{M}|$. Therefore, we have Theorem 5.4. \square

Ideal Functionality: $\mathcal{F}_{\text{pake}}$
<p><u>Session Initialization:</u> P_i inputs $(\text{NewSession}, \text{ssid}, \text{pw}, P_i, P_j)$.</p> <p>// This interface allows a user to initialize a session to a server using pw.</p> <ul style="list-style-type: none"> – If this is the first NewSession query, or if it is the second NewSession query and there is a record $(\text{NewSession}, P_j, P_i, \text{pw}', *) \in \mathcal{L}$, then record $(\text{ssid}, P_i, P_j, \text{pw}, \text{fresh}) \in \mathcal{L}$. – Sends $(\text{NewSession}, \text{ssid}, P_i, P_j)$ to \mathcal{S}
<p><u>Key Generation:</u> \mathcal{S} inputs $(\text{NewKey}, \text{ssid}, P_i, \text{SK})$, where $\text{SK} \in \mathcal{SK}$.</p> <p>// This interface allows parties to have a session key consistent with the state of their records. If \mathcal{S} performed successfully active attack on the password of P_i and ssid, then this query allows it to set up the session key on its choice. Otherwise, this query only sets up a random session key for ssid.</p> <ul style="list-style-type: none"> – If there is a record of the form $(\text{ssid}, P_i, P_j, \text{pw}, \text{status}) \in \mathcal{L}$, for any value status, and this is the first NewKey query for P_i. <ol style="list-style-type: none"> 1. If $\text{status} = \text{compromised}$, or if one of the parties P_i or P_j is corrupted, then send (ssid, SK) to P_i; 2. If $\text{status} = \text{fresh}$ and there is a record $(\text{ssid}, P_j, P_i, \text{pw}', \text{status}')$ with $\text{pw}' = \text{pw}$, and a session key SK' has been set to P_j, that was fresh at that time, then send $(\text{ssid}, \text{SK}')$ to P_i; 3. Else, choose a random key $\text{SK}' \xleftarrow{\\$} \mathcal{SK}$ and send $(\text{ssid}, \text{SK}')$ to P_i. – Update the record as completed.
<p><u>Active attack:</u> \mathcal{S} inputs $(\text{TestPW}, \text{ssid}, P_i, \text{pw}')$.</p> <p>// This query model online dictionary attacks.</p> <ul style="list-style-type: none"> – If there exists record $(\text{ssid}, P_i, P_j, \text{pw}, \text{fresh}) \in \mathcal{L}$, do: <ol style="list-style-type: none"> 1. If $\text{pw} = \text{pw}'$, then mark the record as compromised and reply to \mathcal{S} with "correct". 2. If $\text{pw} \neq \text{pw}'$, then mark the record as interrupted and reply to \mathcal{S} with "wrong".

Figure 24: Functionality $\mathcal{F}_{\text{pake}}$

C On the Universal Composability of Π

In this section, we discuss how to extend the BPR security proof in Section 4.1 to prove the universal composability (UC) of Π . We use the PAKE functionality $\mathcal{F}_{\text{pake}}$ in [BCP⁺23, Figure 3]. For simplicity, we only give the description of $\mathcal{F}_{\text{pake}}$ (Figure 24) and only consider static corruptions. Please refer [BCP⁺23] for full details and adaptive corruptions model.

Informally, to prove Π securely emulates $\mathcal{F}_{\text{pake}}$, we need to show that, for any PPT adversary \mathcal{A} , there exists a simulator such that for any PPT environment \mathcal{Z} , the advantage of \mathcal{Z} that distinguishes whether it is interacting with \mathcal{A} in the real world or it is interacting with \mathcal{S} in the ideal world is negligible. In the ideal world, \mathcal{S} only interacts with \mathcal{Z} and $\mathcal{F}_{\text{pake}}$.

Normally, to prove the UC security, we need to construct a simulator \mathcal{S} that can simulate a real world game for the adversary \mathcal{A} by only using the information from the ideal functionality. Since ideal functionality only has trivial information (in most cases), \mathcal{S} simulates a real world game means that running the protocol in the real world does not reveal too much meaningful information to the adversary (and thus the protocol is secure). For full understanding of universal composability, please refer to [Can01].

Now we sketch how to extend our BPR proof to a UC one. Give an adversary \mathcal{A} , our goal is to construct a simulator \mathcal{S} that, it only uses the information from $\mathcal{F}_{\text{pake}}$ to simulate a real world game for \mathcal{A} . Actually, the game \mathbf{G}_{12} in Figure 21 has a similar idea. In \mathbf{G}_{12} , the game simulator simulates all oracles without knowing passwords (unless \mathcal{A} issues CORRUPT queries or outputs final bit b'). We construct a simulator \mathcal{S} that has some similar behaviors with \mathbf{G}_{12} .

- \mathcal{S} simulates $\text{H}, \text{IC}_1 = (\text{E}_1, \text{D}_1)$, and $\text{IC}_2 = (\text{E}_2, \text{D}_2)$ as in \mathbf{G}_{12} in Figure 21. Specifically, if a fresh query to D_1 will generate a key pair (pk, sk) , as we did in Lemma 4.4.
- Upon receiving $(\text{NewSession}, \text{ssid}, \text{U}, \text{S})$ from $\mathcal{F}_{\text{pake}}$, where (U, S) is a user-server pair: \mathcal{S} sends a fresh e_1 to \mathcal{A} and records it. This is similar to the simulation of SENDINIT in Figure 21.
- Upon server S receiving e_1 from \mathcal{A} on behalf of U with session id ssid . \mathcal{S} does the following checks:

- If there exists pw such that $(\text{pw}, \text{pk}, e_1, \text{enc}) \in \mathcal{L}_1$, \mathcal{S} sends $(\text{TestPW}, \text{ssid}, \text{S}, \text{pw})$ to $\mathcal{F}_{\text{pake}}$. If $\mathcal{F}_{\text{pake}}$ replies "correct", then this means that \mathcal{S} performs a successful online attack and recover $\text{pw}_{\text{U,S}}$. This corresponds to the event that $\text{Guess}_{\text{ser}}$ is triggered in Figure 14. In this case, \mathcal{S} honestly generates e_2 and computes the session key SK as we did in Lines 25 to 31 in Figure 14. Then, \mathcal{S} sends e_2 to \mathcal{A} and sends $(\text{NewKey}, \text{ssid}, \text{S}, \text{SK})$ to $\mathcal{F}_{\text{pake}}$ (which can correctly set up the session key).
 - In other cases, then \mathcal{S} sends $(\text{NewKey}, \text{ssid}, \text{S}, \text{SK})$ to $\mathcal{F}_{\text{pake}}$, where SK is sampled uniformly at random.
- Upon user U receiving e_2 from \mathcal{A} on behalf of S with session id ssid , \mathcal{S} does the following:
 - If there exists pw such that $(\text{pw}, \text{c}, e_2, \text{enc}) \in \mathcal{L}_1$, \mathcal{S} sends $(\text{TestPW}, \text{ssid}, \text{U}, \text{pw})$. If $\mathcal{F}_{\text{pake}}$ replies "correct", then this means that \mathcal{S} correctly guesses $\text{pw}_{\text{U,S}}$. This corresponds to the event that $\text{Guess}_{\text{user}}$ is triggered in Figure 14. In this case, \mathcal{S} honestly computes the session key SK as we did in Lines 67 to 75. Then, \mathcal{S} sends $(\text{NewKey}, \text{ssid}, \text{U}, \text{SK})$ to $\mathcal{F}_{\text{pake}}$ (which can correctly set up the session key). Here we need to use the argument in the proof of Theorem 4.1 to show that at most one such pw exists in $\in \mathcal{L}_1$
 - In other cases, then \mathcal{S} sends $(\text{NewKey}, \text{ssid}, \text{U}, \text{SK})$ to $\mathcal{F}_{\text{pake}}$, where SK is sampled uniformly at random.

This is a very high-level description. The key idea here is that we translate the checking of $\text{Guess}_{\text{user}}$ and $\text{Guess}_{\text{ser}}$ to active attacks on $\mathcal{F}_{\text{pake}}$. We believe that we can *tightly* construct such simulator by our proof technique used in Theorem 4.1.