Lattice-based Authenticated Key Exchange with Tight Security*

Jiaxin Pan [1] Benedikt Wagner [2,3] Runzhi Zeng [1]

June 2, 2023

[1] Department of Mathematical Sciences,
NTNU – Norwegian University of Science and Technology, Trondheim, Norway
jiaxin.pan@ntnu.no, runzhi.zeng@ntnu.no
[2] CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
benedikt.wagner@cispa.de
[3] Saarland University, Saarbrücken, Germany

Abstract

We construct the first tightly secure authenticated key exchange (AKE) protocol from lattices. Known tight constructions are all based on Diffie-Hellman-like assumptions. Thus, our protocol is the *first* construction with tight security from a post-quantum assumption.

Our AKE protocol is constructed tightly from a new security notion for key encapsulation mechanisms (KEMs), called one-way security against checkable chosen-ciphertext attacks (OW-ChCCA). We show how an OW-ChCCA secure KEM can be tightly constructed based on the Learning With Errors assumption, leading to the desired AKE protocol. To show the usefulness of OW-ChCCA security beyond AKE, we use it to construct the *first* tightly bilateral selective-opening (BiSO) secure PKE. BiSO security is a stronger selective-opening notion proposed by Lai et al. (ASIACRYPT 2021).

Keywords: Authenticated key exchange, lattices, tight security, selective-opening security, random oracle.

1 Introduction

Authenticated key exchange (AKE) protocols enable two parties to securely exchange a session key and establish a secure channel. As a crucial building block for secure communication, its security needs to be carefully proven. Compared to many other cryptographic primitives, security proofs of AKE protocols are often very complicated, mostly because an active adversary against an AKE protocol has very strong yet realistic capabilities. For instance, it can control the communication in public networks and arbitrarily modify messages transferred there. Furthermore, it can corrupt parties' secret keys, reveal session keys or even internal states, while adaptively attacking other "fresh" sessions in a meaningful manner. These capabilities are formally captured by security models, such as the Bellare-Rogaway [BR94] and the (extended) Canetti-Krawczyk [CK01, LLM07] models.

Tightness. The strong and complex security requirements do not only make it difficult to prove AKE security, but also introduce a large security loss. The security loss quantitatively measures the gap between the concrete security of a cryptographic protocol and the hardness of the underlying assumption. More precisely, in the security proof, we show that the underlying assumption P implies the security of a cryptographic protocol Π, and establish a relation $\varepsilon_\Pi \leq \ell \cdot \varepsilon_P$ between the advantage of attacking Π and breaking P, where ℓ is called the security loss. We call a proof *tight*, when ℓ is a small constant

*The work of Pan and Zeng is supported by the Research Council of Norway under Project No. 324235. Parts of the work were done while the second author was visiting NTNU. The visit was supported by the same project.

independent of parameters unknown at deployment time such as numbers of parties, protocol sessions, signatures, etc.. We do not distinguish full tightness (i.e., ℓ being a small constant) and almost tightness (i.e., ℓ being linearly dependent on the security parameter). Instead, we are precise with the concrete security loss.

Unfortunately, most of existing AKE protocol are non-tight and, in particular, come with a security loss significantly larger than for other primitives. More precisely, such a protocol often loses a *quadratic* factor in the number of all sessions established in the protocol's lifetime, while a non-tight signature scheme may only lose a linear factor in the number of all issued signatures. Considering today's massive amount of TLS connections, this quadratic security loss is too large to be compensated in practice, since increasing parameters may lead to an intolerable performance overhead. Even if increasing security parameters is an option, it is impossible to correctly guess parameters such as the number of all protocol sessions, since they are unknown at the time of deployment. If our estimation is too small, the provided security guarantee is not backed by the security proof. If our estimation is too large, we end up with an unnecessarily inefficient implementation.

As a result, tightly secure AKE protocols have become an active area recently. Results include feasibility [BHJ+15, LLGW20, HJK+21], practical constructions [GJ18, JKRS21], and concrete analysis of deployed protocols [DJ21, DG21, PQR21]. All these works require techniques based on variants of the Diffie-Hellman assumption. Currently, there is no tightly secure AKE protocol based on a post-quantum assumption.

Our Goal. Our goal is to construct a lattice-based AKE protocol with tight security. We consider a multi-challenge setting defined by the "Single-Bit-Guess" (SBG) security model, where an adversary is given multiple challenge session keys and all the challenge keys are either real or random depending on a single bit. Another multi-challenge notion is the "Multi-Bit-Guess" (MBG) model where the distribution of each session key is decided by a different random bit. As pointed out by Jager et al. [JKRS21], the SBG model is more meaningful than the MBG model, and it can be composed tightly with symmetric primitives to yield a secure channel, while this is not known for the MBG model.

Limitations of Existing Approaches. We survey existing approaches in tightly secure AKE in the SBG model and their limitations in achieving our goal:

Strong DH-based Approaches. Diemert and Jager [DJ21] and, independently, Davis and Günther [DG21] gave tight security proofs of the three-message TLS 1.3 handshake AKE protocol with explicit authentication. The two-message protocol of Pan et al. [PQR21] also falls into this category. All their protocols are (or are similar to) signed Diffie-Hellman protocol and their tight security proofs are all based on the Strong Diffie-Hellman (StDH) assumption [ABR01] and the multi-user security of digital signatures. First of all, we do not have a StDH-like assumption in the lattice setting. This seems inherent, since the gap between decisional and computational variants of an assumption does not exist for lattices. For instance, the decisional Learning With Errors (LWE) assumption [Reg05] is equivalent to its computational version. Secondly, the signature scheme of Pan and Wagner [PW22] is the only known lattice-based scheme with tight multi-user security. Although its signature size is compact and independent of the message length, it is still not efficient, due to the use of OR-proof techniques. The inefficiency of signature schemes can make the resulting AKE protocols impractical.

HPS-based Approaches. Jager et al. [JKRS21] proposed a very efficient tightly secure AKE protocol in the SBG model. Moreover, its security model supports internal state reveals from the adversary. Their construction follows the generic "KEM-to-AKE" transformation [HKSU20] with a multi-receiver non-committing key encapsulation mechanism (KEM), and this KEM is only known to be constructed tightly based on the number-theoretic hash proof systems (HPS). A follow-up work of Han et al. [HJK+21] also relies on number-theoretic HPS and a multi-user secure signature scheme. Both works require tight random self-reducibility of the subset membership problem in the HPS.

Existing lattice-based HPS [KV09, ZY17, BBDQ18] do not have suitable properties to tightly implement frameworks in [JKRS21, HJK+21]. For instance, frameworks in [JKRS21, HJK+21] require tight random self-reducibility, but constructions in [KV09, ZY17] do not have this property, since their language instances are associated with some labels and cannot be easily re-randomized. Another undesirable property is the approximate correctness of the lattice-based HPS. Similar to the password-based AKE [KV09, ZY17, BBDQ18], it is highly non-trivial whether approximate HPS can be fit in the AKE frameworks as in [JKRS21, HJK+21]. Finally, existing lattice-based

HPS are very inefficient. For instance, the construction in [BBDQ18] has only one-bit hash values, and extending it to many-bit, which is necessary for security, requires expanding the public key per portion to the number of hash bits. The resulting AKE protocol is very inefficient. For Han et al.'s protocol, the efficiency is even worse, due to the inefficiency of Pan-Wagner's signature scheme.

1.1 Our Contributions

We construct the first tightly secure lattice-based AKE protocol in the random oracle model (ROM). Its security is based on the decisional Learning With Errors (LWE) assumption with security loss that is independent of parameters such as the number of users or protocol sessions. Our protocol is a two-pass implicit AKE, and it does not require any signature. We use the multi-challenge AKE security model as in [JKRS21], namely, it considers the SBG security and allows an adversary to adaptively corrupt long-term secret keys, reveal session keys and internal states, and make multiple Test queries whose outputs are the non-comprised session keys or random keys. This model captures key compromise impersonation and reflection attacks, and weak forward secrecy, which is the strongest forward secrecy a two-pass implicit AKE protocol can have [Kra05].

Tight AKE from One-Way Checkable CCA Security. Our approach is modular, summarized in Figure 2. To enable tight security from lattices, we introduce a new security notion for KEMs, called one-way checkable security against chosen-ciphertext attacks (OW-ChCCA) in the multi-user, multi-challenge setting. This new notion is sufficient to construct a tightly secure AKE protocol, and can be constructed efficiently and tightly from the LWE assumption. In a nutshell, it is a multi-user, multi-challenge enhancement of one-way security against plaintext-checkable attacks (OW-PCA) [OP01]. More precisely, in our OW-ChCCA security, adversaries are given multiple challenge ciphertexts and multiple users' public keys. Adversaries can check whether a pair of encapsulated key and ciphertext is valid wrt some user via a Check oracle, and are allowed to corrupt some of the user secret keys. Different to OW-PCA, an adversary can additionally decapsulate any ciphertexts, including the challenge ciphertexts. This decapsulation is stronger than the decapsulation of the standard CCA notion where only non-challenge ciphertexts can be decapsulated. To highlight this capability, we model decapsulating challenge ciphertexts as an additional oracle Reveal in our definition. Our OW-ChCCA security guarantees that it is still hard for such an adversary to decapsulate the remaining ciphertexts on its own.

We propose two different approaches to construct tightly secure AKE protocols from an OW-ChCCA KEM in the ROM. Our first approach (cf. Section 4.1) is a generic construction of AKE protocols directly from an OW-ChCCA secure KEM. Our second approach (cf. Section 5.1) is to firstly show that our OW-ChCCA security tightly implies a non-committing KEM (NCKEM) as defined in [JKRS21]. Then via the generic transformation in [JKRS21], this yields a tight AKE protocol. This is less direct than our first approach, and each user needs to do some additional hashing, compared to the first approach, due to the "OW-ChCCA-to-NCKEM" transformation. Figure 1 gives an overview of these two approaches.

Our motivation of OW-ChCCA is to "outsource" all necessary properties for tight AKE into a notion for KEMs. We think that this is easier than directly constructing a tightly secure AKE. Conceptually, AKE is quite a complex primitive, as it is interactive and adversaries can inject new messages adaptively. If we directly construct a tightly secure AKE from lattices, it would be very difficult to start, since there are many corner cases we need to handle because of the complex adversary strategy (which is reflected by the freshness definition). Our OW-ChCCA notion simplifies this complex task. Moreover, we think that OW-ChCCA is more natural and easier to understand than NCKEM as in [JKRS21]. For instance, the NCKEM is defined wrt. a random oracle per user.

OW-ChCCA from LWE, Tightly. Our second step is constructing an efficient OW-ChCCA secure KEM tightly from the LWE assumption in the ROM. The technical challenge is to design a scheme such that its security reduction can tightly embed the LWE problem challenge, while being able to respond all oracle queries defined by OW-ChCCA, in particular, the secret key corruption (Corr) and challenge ciphertext decapsulation (Reveal). Our construction has a novel use of the Naor-Yung (NY) double encryption [NY90] and the lossy LWE approach as in [LSSS17, KYY18] to resolve this. Different to the NY encryption, we do not require non-interactive zero-knowledge proofs by using random oracles and carefully programming them. Our concrete scheme and proof are rather technical. We refer to Section 3.2 for more discussion. Additionally, our idea can be implemented using the Matrix Decisional Diffie-Hellman (MDDH) assumption [EHK+13], which is a generalization of the (standard) DDH assumption, and we

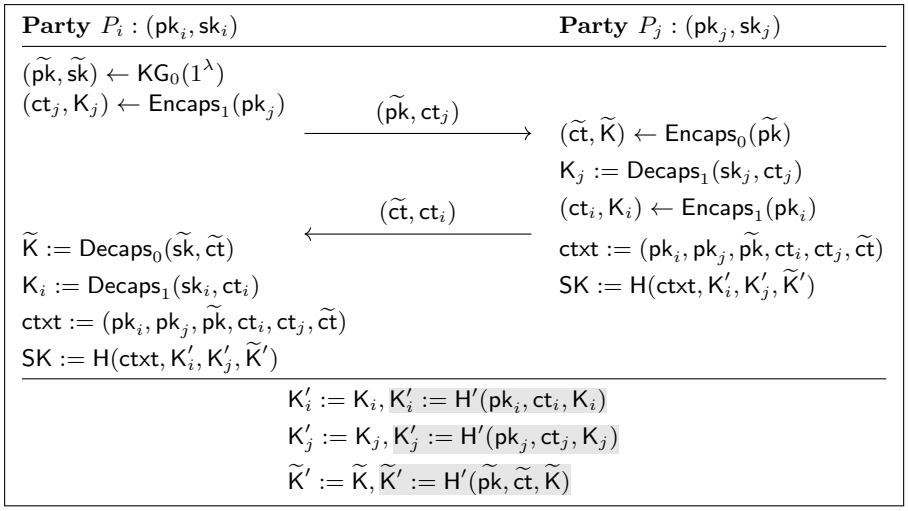

Figure 1: Our two approaches of constructing tightly secure AKE protocols between two parties from OW-ChCCA secure KEMs, $\mathsf{KEM}_1 = (\mathsf{Setup}_1, \mathsf{KG}_1, \mathsf{Encaps}_1, \mathsf{Decaps}_1)$ and $\mathsf{KEM}_0 = (\mathsf{Setup}_0, \mathsf{KG}_0, \mathsf{Encaps}_0, \mathsf{Decaps}_0)$. Our two approaches only differ on how the final session keys are derived. We mark the difference in our second approach with gray. H and H' are two independent hash functions.

construct a OW-ChCCA KEM from MDDH[1].

Efficiency of Our Lattice-based AKE. Asymptotically, our ciphertext is at most twice as long as that of plain Regev's KEM [Reg05]. This carries to our AKE protocol. We argue that such a price is worthy of paying, since it provides stronger theoretically sound security guarantees. A common construction for AKE from lattices is using the generic construction based on a passively secure KEM as in [HKSU20]. Due to the guessing proof strategy, it has a security loss $O((N+S)S \cdot T)$ in the multi-Test setting[2], where N, S, and T are the numbers of users, total sessions, and Test sessions, respectively. We assume $S \approx T$, since an adversary can ask Test queries for some constant fraction of the total sessions. With real-world scenarios, (N, S) can easily reach $(2^{16}, 2^{16})$ (which is the "small-to-medium" scale as in [GJ18, CCG+19]). This means the resulting non-tight AKE (implementing with 128-bit secure Regev's KEM) has 80-bit security supported by the non-tight proof, while ours still has 120-bit security. In the truly large scale with $(N, S) = (2^{32}, 2^{32})$, the non-tight protocol has 32-bit security, while ours still has 120-bit security. 80-bit or 32-bit security is not a secure margin against today's computers.

Beyond AKE: Tight Bilateral Selective-Opening Security. We show that our KEM security notion is useful beyond AKE protocols. One of the examples is simulation-based bilateral selective-opening (SIM-BiSO-CCA) security [LYHW21] for public-key encryption (PKE) schemes. Combining our OW-ChCCA KEM with a one-time pad and a message authentication code, we construct the *first* tightly SIM-BiSO-CCA secure PKE that can be instantiated based on the LWE or MDDH assumption. Informally, selective-opening (SO) security captures the fact that adversaries can learn some randomness used in the encryption algorithm, and bilateral SO security additionally allows user secret key corruptions and is stronger than SO security. SIM-BiSO-CCA formalizes this in a simulation-based manner. This security notion is motivated by some real-world scenarios, where it is expensive to erase cryptographic secrets and adversaries can learn senders' encryption randomness and receivers' secret keys. Currently, the only known SIM-BiSO-CCA PKE is a non-tight scheme in the ROM [LYHW21].

(Tight) Relations to Other KEM Notions. We first observe that by a guessing strategy one can show the standard IND-CCA security non-tightly implies OW-ChCCA with a loss of $O(N \cdot C)$, where N and C are numbers of users and ciphertexts, respectively. We also show that, by hashing its encapsulated key, a OW-ChCCA secure KEM is tightly indistinguishable against enhanced chosen-ciphertext attacks (IND-ECCA) in the ROM. IND-ECCA is a notion proposed by Han et al. [HLG21] to rule out constructing

[1] We believe that the MDDH-based construction in [JKRS21] can also satisfy our notion. As it does not satisfy the deterministic ciphertext derivation property that we need for SIM-BiSO-CCA security, we decided not to present it.

[2] This security loss can be derived as in [HKSU20, Theorem 3] by ignoring the quantum RO and the additive negligible terms. The single-to-multi-challenge reduction introduces the multiplicative term T.

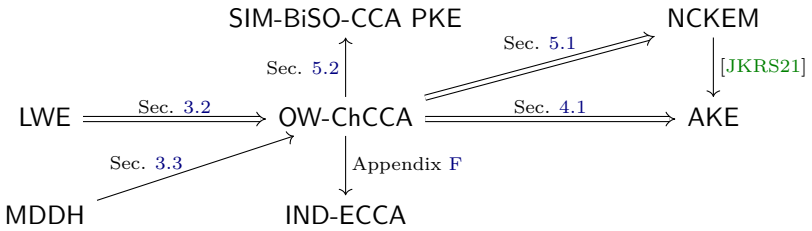

Figure 2: Overview of our contributions. All implications are tight, and they are all new and proposed in this paper except for NCKEM $\rightarrow$ AKE. We highlight those key implications for a tightly secure lattice-based AKE with double arrows, "$\Rightarrow$".

tightly secure AKE from many well-known KEMs in the standard model. Our work bypasses their impossibility result using random oracles. Although an IND-ECCA KEM contains necessary requirements of a secure AKE protocol, there is no formal proof showing that an IND-ECCA KEM tightly implies an AKE. Since IND-ECCA implies the standard IND-CCA security, our OW-ChCCA tightly implies the standard IND-CCA security in the ROM. Combining with our previous discussion, this shows that our OW-ChCCA notion is the core for tight security of different KEM notions, PKE, and AKE. Figure 2 summarizes all our contributions and implications in this paper.

Open Problems. We initiate the first step in constructing efficient lattice-based tightly secure AKE protocols. There are several interesting directions to explore. One of them is to construct a OW-ChCCA secure KEM with shorter ciphertexts from lattices, which will lead to more efficient tightly secure AKE. Although we consider an AKE from post-quantum assumptions in this paper, we are interested in "lifting" our results to the quantum random oracle model. Finally, we are interested in constructing tightly secure lattice-based AKE in the non-programmable random oracle model or even the standard model.

More Post-Quantum AKE. We note that the isogeny-based protocol of de Kock et al. in [dKGV21] is not tight based on a variant of the Commutative Supersingular Isogeny Diffie-Hellman (CSIDH) assumption [CLM+18] and loses a factor in the number of users. Such a non-tight factor is optimal and unavoidable.

2 Preliminaries

A (probabilistic) algorithm $\mathcal{A}$ is PPT, if its running time (denoted by $\mathbf{T}(\mathcal{A})$) can be bounded by a polynomial in its input length. A function $f : \mathbb{N} \rightarrow \mathbb{R}$ with input λ is negligible in λ if $f \in \lambda^{-\omega(1)}$. The term negl denotes a negligible function. A function of the form $1 - \mathsf{negl}(\lambda)$ is said to be overwhelming. Let $\mathcal{A}$ be an algorithm. We write $y \leftarrow \mathcal{A}(x)$ to indicate that y is set to the output of $\mathcal{A}$ in input x with freshly sampled random coins. We write $y := \mathcal{A}(x; \tau)$ to make these coins τ explicit. We write $y \in \mathcal{A}(x)$ to state that y is a potential output (i.e. there are random coins) of $\mathcal{A}$ in input x. For distribution D, we write $x \leftarrow D$ if x is sampled according to D. We write $x \xleftarrow{\$} S$ if x is sampled uniformly from a finite set S. We use the notation $[L] := \{1, \ldots, L\}$ for the first L natural numbers. We use both verbal descriptions and pseudocode to describe games. For that, we make the convention that all variables are initialized to 0, $\perp$ or $\emptyset$, depending on the data type. Also, when we say that the game aborts, this means that the entire game terminates. This is different from the case where an algorithm or oracle outputs $\perp$, in which the game continues. If $\mathbf{G}$ is a game, the notation $\mathbf{G} \Rightarrow b$ indicates that $\mathbf{G}$ outputs b. For all security notions in the multi-user setting, we implicitly assume that the number of users N is polynomially bounded in λ.

Key Encapsulation Mechanisms. We recall the syntax of key encapsulation mechanisms, and give a definition of ciphertext entropy. As introducing a new security notion is part of our contribution, we do not define security here.

Definition 2.1 (Key Encapsulation Mechanism). A key encapsulation mechanism is a tuple of algorithms $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encap}, \mathsf{Decap})$ where $\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encap}$ are PPT and Decap is deterministic:

- $\mathsf{Setup}(1^\lambda) \rightarrow \mathsf{par}$ takes as input the security parameter 1^λ, and outputs global system parameters par. We assume that par implicitly define ciphertext space $\mathcal{C} = \mathcal{C}_{\mathsf{par}}$, key space $\mathcal{K} = \mathcal{K}_{\mathsf{par}}$, and public key space $\mathcal{P} = \mathcal{P}_{\mathsf{par}}$.

- $\mathsf{Gen}(\mathsf{par}) \to (\mathsf{pk}, \mathsf{sk})$ takes as input parameters par, and outputs a public key $\mathsf{pk} \in \mathcal{P}$ and a secret key sk.
- $\mathsf{Encap}(\mathsf{pk}) \to (\mathsf{ct}, K)$ takes as input a public key $\mathsf{pk} \in \mathcal{P}$, and outputs a ciphertext $\mathsf{ct} \in \mathcal{C}$, and a key $K \in \mathcal{K}$.
- $\mathsf{Decap}(\mathsf{sk}, \mathsf{ct}) \to K$ is deterministic, takes as input a secret key sk and a ciphertext $\mathsf{ct} \in \mathcal{C}$, and outputs a key $K \in \mathcal{K} \cup \{\perp\}$.

We say that KEM is ρ-correct, if for every $\mathsf{par} \in \mathsf{Setup}(1^\lambda)$, the following probability is at least ρ:

$$\Pr\left[K = K' \mid (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{par}), (\mathsf{ct}, K) \leftarrow \mathsf{Encap}(\mathsf{pk}), K' \leftarrow \mathsf{Decap}(\mathsf{sk}, \mathsf{ct})\right].$$

For some constructions, we may require that the scheme has high public key or ciphertext entropy. We give formal definitions of these natural notions in Appendix A.

Background on Lattices. Let Λ be an m-dimensional lattice, i.e. a discrete additive subgroup of $\mathbb{R}^m$. For $s > 0$, we define $D_{\Lambda,s}$ to be the distribution proportional to $\rho_s(\mathbf{x}) := \exp(-\pi\|\mathbf{x}\|^2/s^2)$ restricted to Λ. The distribution $D_{\Lambda,s}$ is called the discrete Gaussian distribution with parameter s over Λ. Further, we make the convention that elements in $\mathbb{Z}_q$ are represented by their representative from $\{-(q-1)/2, \ldots, (q-1)/2\}$ (if q is odd) or $\{-q/2+1, \ldots, q/2\}$ (if q is even). We use some standard facts about discrete Gaussians, see [MR04, GPV07].

Lemma 2.2 *Let $n, m \in \mathbb{N}$. Let q be a prime at least polynomial in n and $m \geq 2n \log q$. Consider any $\omega(\sqrt{\log m})$ function and $s \geq \omega(\sqrt{\log m})$. Then for all but a negligible (in n) fraction of all $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ the following distribution is statistically close to uniform over $\mathbb{Z}_q^n$: $\{\mathbf{Ae} \mid \mathbf{e} \leftarrow D_{\mathbb{Z}^m,s}\}$.*

Lemma 2.3 *Consider any $\omega(\sqrt{\log m})$ function and $s \geq \omega(\sqrt{\log m})$. Then we have*

$$\Pr\left[\|\mathbf{x}\| > s\sqrt{m} \mid \mathbf{x} \leftarrow D_{\mathbb{Z},s}^m\right] \leq 2^{-m+1}.$$

We also use the following lemma about the lossiness of a certain matrix distribution, following [AKPW13, KYY18]. It lower bounds the so called "smooth average min-entropy" $\tilde{H}_\infty^\cdot(\cdot|\cdot)$ [KYY18].

Lemma 2.4 *Let n, k, m, q, η be positive integers, $\beta, \alpha' > 0$ such that $\alpha' \geq \beta\eta nmq$. Let χ be a distribution such that $\Pr[|x| \geq \beta q \mid x \leftarrow \chi] \leq \mathsf{negl}(\lambda)$. Let $\mathbf{s}$ be uniformly distributed over $[-\eta, \eta]^n$, and $\mathbf{e}$ be distributed according to $D_{\mathbb{Z},\alpha'}^m$. Let $\mathbf{A} := \bar{\mathbf{A}}\mathbf{C} + \mathbf{F}$ for*

$$\bar{\mathbf{A}} \xleftarrow{\$} \mathbb{Z}_q^{n \times k}, \ \mathbf{C} \xleftarrow{\$} \mathbb{Z}_q^{k \times m}, \ \mathbf{F} \leftarrow \chi^{n \times m}.$$

Then, for any $\epsilon \geq 2^{-\lambda}$, we have

$$\tilde{H}_\infty^\epsilon\left(\mathbf{s} \middle| \mathbf{A}^t\mathbf{s} + \mathbf{e}\right) \geq H_\infty(\mathbf{s}) - (k + 2\lambda)\log q - \mathsf{negl}(\lambda).$$

We make use of the generalized leftover hash lemma, taken from [KYY18].

Lemma 2.5 *Let $\mathcal{H} := \{h_k : \mathcal{X} \to \mathcal{Y}\}_k$ be a universal family of hash functions, where keys k are distributed according to a distribution K. Let U denote a random variable distributed uniformly over $\mathcal{Y}$. Let X be random variable with values in $\mathcal{X}$ and I be any random variable. Let $\epsilon \geq 0$. Then, the statistical distance between $(K, h_K(X), I)$ and (K, U, I) is at most*

$$2\epsilon + \frac{1}{2}\sqrt{2^{-\tilde{H}_\infty^\epsilon(X|I)} \cdot |\mathcal{Y}|}.$$

Our construction relies on the well-known LWE assumption [Reg05, Pei09, BLP+13].

Definition 2.6 (LWE Assumption). Let $n = n(\lambda) \in \mathbb{N}$, $m = m(n) \in \mathbb{N}$, $q = q(n)$ be prime number and $\chi = \chi(n)$ be a distribution over $\mathbb{Z}$. We say that the $\mathsf{LWE}_{n,m,q,\chi}$ assumption holds, if for every PPT algorithm $\mathcal{B}$ the following advantage is negligible in λ:

$$\begin{aligned}\mathsf{Adv}_{\mathcal{B}}^{\mathsf{LWE}_{n,m,q,\chi}}(\lambda) := |&\Pr\left[\mathcal{B}(\mathbf{A}, \mathbf{b}) = 1 \mid \mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}, \mathbf{b} \xleftarrow{\$} \mathbb{Z}_q^m\right] \\ &- \Pr\left[\mathcal{B}(\mathbf{A}, \mathbf{A}^t\mathbf{s} + \mathbf{e}) = 1 \mid \mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}, \mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^n, \mathbf{e} \leftarrow \chi^m\right]|.\end{aligned}$$

3 One-Way Checkable CCA Security

We first propose a new security notion for key encapsulation mechanisms (KEM), One-Way Checkable Security against Chosen-Ciphertext Attacks (OW-ChCCA). Then we realize this notion with lattices in a tight way.

3.1 Definition of OW-ChCCA Security

Before we formalize our notion for KEMs, we give some intuitions behind it. At a very high level, our OW-ChCCA security can be seen as an extension of OW-PCA security [OP01]. Our goal here is to formalize the "minimal" requirements on KEM for constructing tightly secure AKE. We first observe that an indistinguishability notion, such as IND-CPA, is not necessary for AKE, and the weaker, one-way notion (namely, decapsulating the challenge ciphertexts) is sufficient. This is because the AKE session keys are often derived from the corresponding KEM key using a hash function which is modeled as a random oracle (RO). By searching the RO-history, it can be shown that a one-way notion tightly implies the corresponding IND one.

To tightly use a one-way notion in our security reduction without guessing, we still need to identify the correct key-ciphertext pairs. Thus, we provide an oracle to check if a key-ciphertext pair is valid, which essentially results in the OW-PCA notion for KEMs. However, OW-PCA is not sufficient to get authenticated key exchange tightly. At a high level, an AKE adversary can adaptively attack multiple sessions and forces the reduction to guess which are the challenge sessions (aka. Test sessions). This will lead to a large security loss. We therefore add additional adversary capabilities (i.e. oracles) to KEM's OW-PCA security to resolve this:

- The adversary can get multiple challenge ciphertexts.
- The adversary has access to a decapsulation oracle, which is used for non-challenge ciphertexts.
- The adversary can adaptively decapsulate some challenge ciphertexts. This allows the AKE security reduction to answer session key reveals for sessions that possibly contain challenge KEM ciphertexts. An AKE adversary can force this happen.
- The adversary can adaptively corrupt users' their secret key. This corresponds to long-term secret key corruptions in the AKE protocol.

We formalize the OW-ChCCA security for KEMs as follows:

Definition 3.1 (OW-ChCCA Security). Let $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encap}, \mathsf{Decap})$ be a key encapsulation mechanism and consider the game OW-ChCCA defined in Figure 3. We say that KEM is OW-ChCCA secure, if for all PPT adversaries $\mathcal{A}$, the following advantage is negligible:

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

For our construction of SIM-BiSO-CCA secure encryption in Section 5.2, we require that KEM has deterministic ciphertext derivation:

Definition 3.2 (Deterministic Ciphertext Derivation). Let $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encap}, \mathsf{Decap})$ be a key encapsulation mechanism with key space $\mathcal{K}$. We say that KEM has deterministic ciphertext derivation, if there is a deterministic algorithm $\widehat{\mathsf{Encap}}$, such that for all $\mathsf{par} \in \mathsf{Setup}(1^\lambda)$ and all $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{Gen}(\mathsf{par})$, the following two distributions are equivalent

$$\{(\mathsf{ct}, K) \mid K \xleftarrow{\$} \mathcal{K}, \quad \mathsf{ct} := \widehat{\mathsf{Encap}}(\mathsf{pk}, K)\} \text{ and } \{(\mathsf{ct}, K) \leftarrow \mathsf{Encap}(\mathsf{pk})\}.$$

3.2 Construction from Lattices

We construct $\mathsf{KEM}_{\mathsf{LWE}}$ that is tightly OW-ChCCA secure under the LWE assumption. Our scheme is described in Figure 4. It uses algorithms SampleD and Round:

- $\mathsf{SampleD}(m, \alpha'; \rho) \to \mathbf{e}$: Sample Gaussian $\mathbf{e} \leftarrow D^m_{\mathbb{Z},\alpha'}$ using random coins $\rho \in \{0,1\}^\lambda$.
- $\mathsf{Round}(\mathbf{t}) \to \mathbf{h}$: Do componentwise rounding of $\mathbf{t} \in \mathbb{Z}_q^\lambda$ to get $\mathbf{h} \in \{0,1\}^\lambda$, i.e. for all $i \in [\lambda]$, we have $\mathbf{h}_i = 0$ if $\mathbf{t}_i$ is closer to 0 than to $\lfloor q/2 \rceil$, and $\mathbf{h}_i = 1$ otherwise.

Game $\mathsf{OW\text{-}ChCCA}^{\mathcal{A}}_{\mathsf{KEM}}(\lambda)$

$\mathsf{par} \leftarrow \mathsf{Setup}(1^\lambda)$
for $i \in [N] : (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(\mathsf{par})$
$\mathrm{O}_1 := (\textsc{Enc}, \textsc{Dec}, \textsc{Reveal})$
$\mathrm{O}_2 := (\textsc{Corr}, \textsc{Check})$
$(i^*, \mathsf{ct}^*, K^*) \leftarrow \mathcal{A}^{\mathrm{O}_1, \mathrm{O}_2}(\mathsf{par}, (\mathsf{pk}_i)_{i \in [N]})$
if $(i^*, \mathsf{ct}^*, K^*) \notin \mathcal{L}_{\textsc{Enc}}$: **return** 0
if $i^* \in \mathcal{L}_{\textsc{Corr}}$: **return** 0
if $(i^*, \mathsf{ct}^*) \in \mathcal{L}_{\textsc{Reveal}}$: **return** 0
return 1

Oracle $\textsc{Corr}(i)$

$\mathcal{L}_{\textsc{Corr}} := \mathcal{L}_{\textsc{Corr}} \cup \{i\}$
return sk_i

Oracle $\textsc{Dec}(i, \mathsf{ct}')$

if $\exists K'$ s.t. $(i, \mathsf{ct}', K') \in \mathcal{L}_{\textsc{Enc}}$: **return** $\perp$
return $K' := \mathsf{Decap}(\mathsf{sk}_i, \mathsf{ct}')$

Oracle $\textsc{Reveal}(i, \mathsf{ct})$

if $\exists K$ s.t. $(i, \mathsf{ct}, K) \in \mathcal{L}_{\textsc{Enc}}$:
  $\mathcal{L}_{\textsc{Reveal}} := \mathcal{L}_{\textsc{Reveal}} \cup \{(i, \mathsf{ct})\}$
  return K
return $\perp$

Oracle $\textsc{Enc}(i)$

$(\mathsf{ct}, K) \leftarrow \mathsf{Encap}(\mathsf{pk}_i)$
$\mathcal{L}_{\textsc{Enc}} := \mathcal{L}_{\textsc{Enc}} \cup \{(i, \mathsf{ct}, K)\}$
return ct

Oracle $\textsc{Check}(i, \mathsf{ct}, K)$

if $\mathsf{Decap}(\mathsf{sk}_i, \mathsf{ct}) = K$:
  return 1
return 0

Figure 3: The game $\mathsf{OW\text{-}ChCCA}$ for a key encapsulation mechanism $\mathsf{KEM} := (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encap}, \mathsf{Decap})$.

Our scheme is parameterized by matrix dimensions $n, m, k \in \mathbb{N}$, a modulus $q \in \mathbb{N}$, and (Gaussian) widths $\alpha, \alpha', \gamma, \eta > 0$. The scheme also makes use of random oracles $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^\lambda$ and $\mathsf{G}\colon \{0,1\}^* \to [-\eta, \eta]^n \times \{0,1\}^\lambda \times \{0,1\}^\lambda \times \{0,1\}^\lambda$. For our analysis, the parameters have to satisfy the following conditions.

- For Lemma 2.2: q prime, $m \geq 2n \log q, \alpha \geq \omega(\sqrt{\log m})$
- For Lemma 2.3: $\alpha, \alpha' \geq \omega(\sqrt{\log m})$
- For Lemmata 2.4 and 2.5: $n \log(2\eta + 1) - (k + 3\lambda) \log q \geq \lambda \log q + \Omega(n)$ and $\alpha' \geq \beta\eta nmq$; we use $\beta q = n$ in Lemma 2.4.
- For correctness: $4\alpha'\alpha m < q$.

For example, given λ, we could conservatively use the following parameter setting.

$$n := 70\lambda \qquad n^6 < q \leq n^7, \qquad \eta := \sqrt{n}, \qquad \gamma := \sqrt{n},$$
$$k := \lambda, \qquad m := 2n \log q, \qquad \alpha := \sqrt{n}, \qquad \alpha' := n^{2.5} m.$$

Correctness follows from standard calculations, and we postpone it to Appendix B. Further, it can easily be seen that $\mathsf{KEM}_{\mathsf{LWE}}$ has deterministic ciphertext derivation.

Before going into the security analysis, we give an overview of the rationale behind our construction, omitting LWE specific details. First, recall that in the security proof, we need to simulate a corruption oracle, returning secret keys sk_i for user i. To do this without using non-tight guessing arguments, we have to know a secret key for each user. As we still need to embed our LWE challenge in the challenge ciphertexts, we should not be able to decrypt them. We can solve this first dilemma by splitting the ciphertext into two parts $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1)$, and having two potential secret keys sk_0 and sk_1, where ct_b allows to recover the encapsulation key K using secret key sk_b. Then, for each user i, we hold sk_{i,b_i} for a random bit b_i. Now, the strategy is to use LWE to modify ct_{1-b_i}. Let us call such modified ciphertext parts inconsistent. Finally, we argue that the adversary does not learn b_i for uncorrupted users, and then we switch roles of b_i and $1 - b_i$ to apply the same argument. This overall strategy can be implemented, if we only have to provide a corruption oracle. However, once we need to simulate a decapsulation oracle, the situation becomes a bit more tricky. Namely, we have to guarantee that decapsulation (simulated using sk_{i,b_i}) does not reveal information about b_i. We solve this challenge by deterministically deriving the ciphertext parts $\mathsf{ct}_0, \mathsf{ct}_1$ from the encapsulated key K. Ciphertext parts could have roughly the form $\mathsf{ct}_b = K \oplus F_b(K)$ for some deterministic F_b. During decapsulation, we recompute $\mathsf{ct}_0, \mathsf{ct}_1$ from K, and only accept K if this recomputation is consistent. A careful analysis shows that this hides all information about b_i. We can also implement a check oracle now using the deterministic functions F_0, F_1. On the other hand, applying changes to $\mathsf{ct}_{1-b_i} = K \oplus F_{1-b_i}(K)$ may lead to circular security problems. At a high level, we solve this issue by implementing the functions F_b using a random oracle.

Alg $\mathsf{Setup}(1^\lambda)$
$\mathbf{return}\ \mathsf{par} := \mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$

Alg $\mathsf{Gen}(\mathsf{par})$
$b \xleftarrow{\$} \{0,1\},\ \mathbf{Z}_b \leftarrow D_{\mathbb{Z},\alpha}^{m\times\lambda}$
$\mathbf{U}_b := \mathbf{A}\mathbf{Z}_b,\ \mathbf{U}_{1-b} \xleftarrow{\$} \mathbb{Z}_q^{n\times\lambda}$
$\mathsf{pk} := (\mathbf{U}_0, \mathbf{U}_1),\ \mathsf{sk} := (\mathbf{Z}_b, b)$
$\mathbf{return}\ (\mathsf{pk}, \mathsf{sk})$

Alg $\mathsf{Encap}(\mathsf{pk})$
$R \xleftarrow{\$} \{0,1\}^\lambda,\ (\mathbf{s}, \rho, \mathbf{h}_0, \mathbf{h}_1) := \mathsf{G}(R)$
$\mathbf{e} := \mathsf{SampleD}(m, \alpha'; \rho)$
$\mathbf{x} := \mathbf{A}^t\mathbf{s} + \mathbf{e}$
$\hat{\mathbf{h}}_0 := \mathbf{U}_0^t\mathbf{s} + \mathbf{h}_0\lfloor q/2\rceil \in \mathbb{Z}_q^\lambda$
$\hat{\mathbf{h}}_1 := \mathbf{U}_1^t\mathbf{s} + \mathbf{h}_1\lfloor q/2\rceil \in \mathbb{Z}_q^\lambda$
$\hat{K}_0 := \mathsf{H}(\mathbf{x}, \hat{\mathbf{h}}_0, \mathbf{h}_0),\quad C_0 := \hat{K}_0 \oplus R$
$\hat{K}_1 := \mathsf{H}(\mathbf{x}, \hat{\mathbf{h}}_1, \mathbf{h}_1),\quad C_1 := \hat{K}_1 \oplus R$
$\mathsf{ct} := (C_0, C_1, \mathbf{x}, \hat{\mathbf{h}}_0, \hat{\mathbf{h}}_1)$
$\mathbf{return}\ (\mathsf{ct}, K := R)$

Alg $\mathsf{Decap}(\mathsf{sk}, \mathsf{ct})$
$\mathbf{let}\ \mathsf{ct} = (C_0, C_1, \mathbf{x}, \hat{\mathbf{h}}_0, \hat{\mathbf{h}}_1)$
$\mathbf{let}\ \mathsf{sk} = (\mathbf{Z}_b, b)$
$\mathbf{h}'_b := \mathsf{Round}(\hat{\mathbf{h}}_b - \mathbf{Z}_b^t\mathbf{x}) \in \{0,1\}^\lambda$
$\hat{K}_b := \mathsf{H}(\mathbf{x}, \hat{\mathbf{h}}_b, \mathbf{h}'_b)$
$R := C_b \oplus \hat{K}_b$
$(\mathbf{s}, \rho, \mathbf{h}_0, \mathbf{h}_1) := \mathsf{G}(R)$
$\mathbf{e} := \mathsf{SampleD}(m, \alpha'; \rho)$
$\hat{\mathbf{h}}'_{1-b} := \mathbf{U}_{1-b}^t\mathbf{s} + \mathbf{h}_{1-b}\lfloor q/2\rceil$
$\hat{K}_{1-b} := \mathsf{H}(\mathbf{x}, \hat{\mathbf{h}}'_{1-b}, \mathbf{h}_{1-b})$
$\mathbf{if}\ \mathbf{x} \neq \mathbf{A}^t\mathbf{s} + \mathbf{e} : \mathbf{return}\ \perp$
$\mathbf{if}\ \hat{K}_{1-b} \oplus R \neq C_{1-b} : \mathbf{return}\ \perp$
$\mathbf{if}\ \mathbf{h}'_b \neq \mathbf{h}_b : \mathbf{return}\ \perp$
$\mathbf{if}\ \hat{\mathbf{h}}'_{1-b} \neq \hat{\mathbf{h}}_{1-b} : \mathbf{return}\ \perp$
$\mathbf{return}\ K := R$

Figure 4: The key encapsulation mechanism $\mathsf{KEM}_{\mathsf{LWE}} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encap}, \mathsf{Decap})$, where $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^\lambda$ and $\mathsf{G}\colon \{0,1\}^* \to [-\eta,\eta]^n \times \{0,1\}^\lambda \times \{0,1\}^\lambda \times \{0,1\}^\lambda$ are random oracles. To save space, one could set $\mathbf{A} := \mathsf{H}^*(0)$ for a random oracle $\mathsf{H}^*\colon \{0,1\}^* \to \mathbb{Z}_q^{n\times m}$.

The next challenge that we have to solve arises from the reveal oracle, which allows to decapsulate challenge ciphertexts. We see that our above strategy is not compatible with this oracle, as we make ciphertexts partially inconsistent, and the adversary can notice this once we reveal the encapsulated key. Again, guessing which ciphertexts will be revealed is not an option without loosing tightness. This means that we have to be able to make inconsistent ciphertext parts consistent again, once a query to the reveal or corruption oracle occurs. We carefully implement this by another use of a random oracle. For simplicity, say we replace $F_b(\cdot)$ by $\mathsf{H}(F_b(\cdot))$, i.e. we have $\mathsf{ct}_{1-b_i} = K \oplus \mathsf{H}(F_{1-b_i}(K))$. Then, to make the ciphertext inconsistent, we set $\mathsf{ct}_{1-b_i} = K \oplus \hat{h}$ for some random $\hat{h}$. To make the ciphertext consistent later on, we program $\mathsf{H}(F_b(K)) := \hat{h}$. It remains to argue that the adversary can not detect the inconsistency by querying $\mathsf{H}(F_b(K))$ before it queries the reveal or corruption oracle. To rule out this bad event, we want to switch $F_b(K)$ to a random element using the LWE assumption. As we have no chance to make such ciphertext consistent again, bounding this bad event requires to define a separate game, which stops once a query to the reveal or corruption oracle occurs. Implemented naively, this leads to a non-tight reduction, applying LWE once per challenge ciphertext. However, it turns out that we can first switch the LWE parameters to lossy mode, as done in [KYY18], and then analyze all of these separate games in a purely statistical way. There is a complex interplay between all of these challenges and potential solutions, and the formal proof requires heavy use of delayed analysis.

Theorem 3.3 *Let* $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^\lambda$ *and* $\mathsf{G}\colon \{0,1\}^* \to [-\eta,\eta]^n \times \{0,1\}^\lambda \times \{0,1\}^\lambda \times \{0,1\}^\lambda$ *be random oracles. If the* $\mathsf{LWE}_{k,m,q,D_{\mathbb{Z},\gamma}}$ *assumption holds, then the scheme* $\mathsf{KEM}_{\mathsf{LWE}}$ *is* $\mathsf{OW\text{-}ChCCA}$ *secure.*

Concretely, for any PPT algorithm $\mathcal{A}$ *there is a PPT algorithm* $\mathcal{B}$ *with* $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B})$ *and*

$$\mathsf{Adv}^{\mathsf{OW\text{-}ChCCA}}_{\mathcal{A},\mathsf{KEM}_{\mathsf{LWE}}}(\lambda) \leq 6n \cdot \mathsf{Adv}^{\mathsf{LWE}_{k,m,q,D_{\mathbb{Z},\gamma}}}_{\mathcal{B}}(\lambda) + \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be a PPT algorithm and $\mathsf{KEM} := \mathsf{KEM}_{\mathsf{LWE}}$. We show the claim using a sequence of games $\mathbf{G}_i$ for $i \in \{0,\ldots,6\}$. To simplify notation, we define

$$\mathsf{Adv}_i := \Pr[\mathbf{G}_i \Rightarrow 1],\ \text{for } i \in \{0,\ldots,6\}.$$

Let us first give an informal overview of the proof strategy. We define $\mathbf{G}_0 := \mathsf{OW\text{-}ChCCA}^{\mathcal{A}}_{\mathsf{KEM}}(\lambda)$ and introduce changes to game $\mathbf{G}_0$ until we end up at game $\mathbf{G}_3$. For this one, we argue that the probability

Game $\mathbf{G}_0$-$\mathbf{G}_6$

$\mathsf{par} := \mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$ // $\mathbf{G}_0$-$\mathbf{G}_1$,$\mathbf{G}_4$-$\mathbf{G}_6$

$\bar{\mathbf{A}} \xleftarrow{\$} \mathbb{Z}_q^{n\times k},\ \mathbf{C} \xleftarrow{\$} \mathbb{Z}_q^{k\times m},\ \mathbf{F} \leftarrow D_{\mathbb{Z},\gamma}^{n\times m},\ \mathsf{par} := \mathbf{A} := \bar{\mathbf{A}}\mathbf{C} + \mathbf{F}$ // $\mathbf{G}_2$-$\mathbf{G}_3$

for $i \in [N]$:

  $b_i \xleftarrow{\$} \{0,1\},\ \mathbf{Z}_{i,b_i} \leftarrow D_{\mathbb{Z},\alpha}^{m\times\lambda},\ \mathbf{U}_{i,b_i} := \mathbf{A}\mathbf{Z}_{i,b_i}$

  $\mathbf{Z}_{i,1-b_i} \leftarrow D_{\mathbb{Z},\alpha}^{m\times\lambda},\ \mathbf{U}_{i,1-b_i} := \mathbf{A}\mathbf{Z}_{i,1-b_i}$ // $\mathbf{G}_5$-$\mathbf{G}_6$

  $\mathbf{U}_{i,1-b_i} \xleftarrow{\$} \mathbb{Z}_q^{n\times\lambda}$ // $\mathbf{G}_0$-$\mathbf{G}_4$

  let $\mathbf{Z}_{i,b_i} = [\mathbf{z}_1 \mid \cdots \mid \mathbf{z}_\lambda]$ // $\mathbf{G}_6$

  if $\exists j \in [\lambda]$ s.t. $\|\mathbf{z}_j\| > \alpha\sqrt{m}$: **abort** // $\mathbf{G}_6$

  let $\mathbf{Z}_{i,1-b_i} = [\mathbf{z}_1 \mid \cdots \mid \mathbf{z}_\lambda]$ // $\mathbf{G}_6$

  if $\exists j \in [\lambda]$ s.t. $\|\mathbf{z}_j\| > \alpha\sqrt{m}$: **abort** // $\mathbf{G}_6$

  $\mathsf{pk}_i := (\mathbf{U}_{i,0}, \mathbf{U}_{i,1}),\ \mathsf{sk}_i := (\mathbf{Z}_{i,b_i}, b_i)$

$(i^*, \mathsf{ct}^*, K^*) \leftarrow \mathcal{A}^{\textsc{Enc},\textsc{Dec},\textsc{Reveal},\textsc{Corr},\textsc{Check},\mathsf{H},\mathsf{G}}(\mathsf{par}, (\mathsf{pk}_i)_{i\in[N]})$

if $(i^*, \mathsf{ct}^*, K^*) \notin \mathcal{L}_{\textsc{Enc}} \vee i^* \in \mathcal{L}_{\textsc{Corr}} \vee (i^*, \mathsf{ct}^*) \in \mathcal{L}_{\textsc{Reveal}}$: **return** 0

return 1

Oracle $\textsc{Enc}(i)$

$R \xleftarrow{\$} \{0,1\}^\lambda$

if $g[R] \neq \bot$: $\mathsf{badR} := 1$, **abort** // $\mathbf{G}_1$-$\mathbf{G}_6$

$(\mathbf{s}, \rho, \mathbf{h}_0, \mathbf{h}_1) := \mathsf{G}(R),\ \mathbf{e} \leftarrow \mathsf{SampleD}(m, \alpha'; \rho)$

$\mathbf{x} := \mathbf{A}^t\mathbf{s} + \mathbf{e},\ \ \hat{\mathbf{h}}_0 := \mathbf{U}_{i,0}^t\mathbf{s} + \mathbf{h}_0\lfloor q/2\rceil,\ \hat{\mathbf{h}}_1 := \mathbf{U}_{i,1}^t\mathbf{s} + \mathbf{h}_1\lfloor q/2\rceil$

$\mathsf{qry}_0 := (\mathbf{x}, \hat{\mathbf{h}}_0, \mathbf{h}_0),\ \mathsf{qry}_1 := (\mathbf{x}, \hat{\mathbf{h}}_1, \mathbf{h}_1)$

if $h[\mathsf{qry}_{b_i}] \neq \bot$: $\mathsf{badK1} := 1$, **abort** // $\mathbf{G}_3$-$\mathbf{G}_6$

if $h[\mathsf{qry}_{1-b_i}] \neq \bot$: $\mathsf{badK0} := 1$, **abort** // $\mathbf{G}_3$-$\mathbf{G}_6$

$\hat{K}_0 := \mathsf{H}(\mathbf{x}, \hat{\mathbf{h}}_0, \mathbf{h}_0),\ \hat{K}_1 := \mathsf{H}(\mathbf{x}, \hat{\mathbf{h}}_1, \mathbf{h}_1)$

$C_0 := \hat{K}_0 \oplus R,\ C_1 := \hat{K}_1 \oplus R$

$\mathsf{ct} := (C_0, C_1, \mathbf{x}, \hat{\mathbf{h}}_0, \hat{\mathbf{h}}_1)$

$\mathcal{L}_R := \mathcal{L}_R \cup \{(R, i, \mathsf{ct})\}$ // $\mathbf{G}_1$-$\mathbf{G}_6$

$\mathcal{L}_{K,0} := \mathcal{L}_{K,0} \cup \{(\mathsf{qry}_{1-b_i}, i, \mathsf{ct})\}$ // $\mathbf{G}_3$-$\mathbf{G}_6$

$\mathcal{L}_{K,1} := \mathcal{L}_{K,1} \cup \{(\mathsf{qry}_{b_i}, i, \mathsf{ct})\}$ // $\mathbf{G}_3$-$\mathbf{G}_6$

$\mathcal{L}_{\textsc{Enc}} := \mathcal{L}_{\textsc{Enc}} \cup \{(i, \mathsf{ct}, K := R)\}$

return ct

Figure 5: The games $\mathbf{G}_0$-$\mathbf{G}_6$ in the proof of Theorem 3.3. Oracles Dec and Check are as in the real games. The remaining oracles are given in Figure 6. Highlighted lines are only executed in the corresponding games.

that $\mathbf{G}_3$ outputs 1 is negligible. Games $\mathbf{G}_i$ for $i \geq 4$ are only needed in Lemmata 3.5 and 3.6, which are used to bound the difference between $\mathbf{G}_0$ and $\mathbf{G}_3$ using a delayed analysis technique. We will now give the details of the proof. The games are formally presented in Figures 5 and 6.

Game $\mathbf{G}_0$: We define $\mathbf{G}_0$ as the real OW-ChCCA game. That is, we set $\mathbf{G}_0 := \mathsf{OW\text{-}ChCCA}_{\mathsf{KEM}}^{\mathcal{A}}(\lambda)$. To fix some notation, we recall how this game works. First, the game samples $\mathsf{par} := \mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$. Then, for each $i \in [N]$, it computes public keys and secret keys $(\mathsf{pk}_i, \mathsf{sk}_i)$ as follows: It samples a bit $b_i \xleftarrow{\$} \{0,1\}$, it samples a matrix $\mathbf{Z}_{i,b_i}$ as in the scheme, and sets $\mathbf{U}_{i,b_i} := \mathbf{A}\mathbf{Z}_{i,b_i}$. Then, it samples $\mathbf{U}_{i,1-b_i}$ uniformly at random as in the scheme. The public key is $\mathsf{pk}_i := (\mathbf{U}_{i,0}, \mathbf{U}_{i,1})$, and the secret key is $\mathsf{sk}_i := (\mathbf{Z}_{i,b_i}, b_i)$. Then, adversary $\mathcal{A}$ gets all public keys and par as input. Further, it gets access to oracles Enc, Dec, Reveal, Corr, Check, as well as random oracles H, G. The game simulates random oracles H, G in a lazy manner, using maps h and g that map the inputs of H, G to the outputs of H, G, respectively. We denote the number of queries to H, G, and Enc by $Q_{\mathsf{H}}, Q_{\mathsf{G}}$, and Q_{E}, respectively. In the end, $\mathcal{A}$ outputs $(i^*, \mathsf{ct}^*, K^*)$. The game outputs 1, if ct^* has been output by $\textsc{Enc}(i^*)$, where it was computed together with K^*, the adversary $\mathcal{A}$ never queried $\textsc{Corr}(i^*)$, and never queried $\textsc{Reveal}(i^*, \mathsf{ct}^*)$. By definition, we have

$$\mathsf{Adv}_0 = \Pr\left[\mathsf{OW\text{-}ChCCA}_{\mathsf{KEM}}^{\mathcal{A}}(\lambda) \Rightarrow 1\right].$$

Game $\mathbf{G}_1$: In game $\mathbf{G}_1$, we introduce a bad event badR, and abort the game if badR occurs. To define the

Oracle $\textsc{Reveal}(i, \mathsf{ct})$

$\mathcal{L}_{\textsc{Reveal}} := \mathcal{L}_{\textsc{Reveal}} \cup \{(i, \mathsf{ct})\}$
$\mathcal{L}_R := \mathcal{L}_R \setminus \{(R, i', \mathsf{ct}') \in \mathcal{L}_R \mid (i', \mathsf{ct}') = (i, \mathsf{ct})\}$ // $\mathbf{G}_1$-$\mathbf{G}_6$
$\mathcal{L}_{K,0} := \mathcal{L}_{K,0} \setminus \{(\mathsf{qry}, i', \mathsf{ct}) \in \mathcal{L}_{K,0} \mid (i', \mathsf{ct}') = (i, \mathsf{ct})\}$ // $\mathbf{G}_3$-$\mathbf{G}_6$
$\mathcal{L}_{K,1} := \mathcal{L}_{K,1} \setminus \{(\mathsf{qry}, i', \mathsf{ct}) \in \mathcal{L}_{K,1} \mid (i', \mathsf{ct}') = (i, \mathsf{ct})\}$ // $\mathbf{G}_3$-$\mathbf{G}_6$
return K
return $\perp$

Oracle $\textsc{Corr}(i)$

$\mathcal{L}_{\textsc{Corr}} := \mathcal{L}_{\textsc{Corr}} \cup \{i\}$
$\mathcal{L}_R := \mathcal{L}_R \setminus \{(R, i', \mathsf{ct}) \in \mathcal{L}_R \mid i' = i\}$ // $\mathbf{G}_1$-$\mathbf{G}_6$
$\mathcal{L}_{K,0} := \mathcal{L}_{K,0} \setminus \{(\mathsf{qry}, i', \mathsf{ct}) \in \mathcal{L}_{K,0} \mid i' = i\}$ // $\mathbf{G}_3$-$\mathbf{G}_6$
$\mathcal{L}_{K,1} := \mathcal{L}_{K,1} \setminus \{(\mathsf{qry}, i', \mathsf{ct}) \in \mathcal{L}_{K,1} \mid i' = i\}$ // $\mathbf{G}_3$-$\mathbf{G}_6$
return sk_i

Oracle $\mathsf{H}(\mathbf{x}', \hat{\mathbf{h}}, \mathbf{h})$

$\mathsf{qry} := (\mathbf{x}', \hat{\mathbf{h}}, \mathbf{h})$ // $\mathbf{G}_3$-$\mathbf{G}_6$
if $\exists i, \mathsf{ct}$ s.t. $(\mathsf{qry}, i, \mathsf{ct}) \in \mathcal{L}_{K,0}$: $\mathsf{badK0} := 1$, **abort** // $\mathbf{G}_3$-$\mathbf{G}_6$
if $\exists i, \mathsf{ct}$ s.t. $(\mathsf{qry}, i, \mathsf{ct}) \in \mathcal{L}_{K,1}$: $\mathsf{badK1} := 1$, **abort** // $\mathbf{G}_3$-$\mathbf{G}_6$
if $h[\mathbf{x}', \hat{\mathbf{h}}, \mathbf{h}] = \perp$: $h[\mathbf{x}', \hat{\mathbf{h}}, \mathbf{h}] \xleftarrow{\$} \{0,1\}^\lambda$
if $\exists (\mathbf{x}'', \hat{\mathbf{h}}', \mathbf{h}') \neq \mathsf{qry}$ s.t. $h[\mathbf{x}'', \hat{\mathbf{h}}', \mathbf{h}'] = h[\mathbf{x}', \hat{\mathbf{h}}, \mathbf{h}]$: **abort** // $\mathbf{G}_6$
return $h[\mathbf{x}', \hat{\mathbf{h}}, \mathbf{h}]$

Oracle $\mathsf{G}(R')$

if $\exists i, \mathsf{ct}$ s.t. $(R', i, \mathsf{ct}) \in \mathcal{L}_R$: $\mathsf{badR} := 1$, **abort** // $\mathbf{G}_1$-$\mathbf{G}_6$
if $g[R'] = \perp$:
$(\mathbf{s}, \rho, \mathbf{h}_0, \mathbf{h}_1) \xleftarrow{\$} \mathbb{Z}_q^n \times \left(\{0,1\}^\lambda\right)^3$
if $\|\mathsf{SampleD}(m, \alpha'; \rho)\| > \alpha' \sqrt{m}$: **abort** // $\mathbf{G}_6$
$g[R'] := (\mathbf{s}, \rho, \mathbf{h}_0, \mathbf{h}_1)$
return $g[R']$

Figure 6: The oracles in the proof of Theorem 3.3. The rest of the games is given in Figure 5. Highlighted lines are only executed in the corresponding games.

event, consider a query of the form $\textsc{Enc}(i)$. Recall that in such a query, the game samples $R \xleftarrow{\$} \{0,1\}^\lambda$ and derives values $\mathbf{s}, \mathbf{e}, \mathbf{h}_0, \mathbf{h}_1$ from it using random oracle G. These are then used to define a ciphertext ct. We say that the bad event badR occurs, if one of the following holds:

- At this point, $\mathcal{A}$ already queried $\mathsf{G}(R)$, or
- at some later point, before any query of the form $\textsc{Corr}(i)$ or $\textsc{Reveal}(i, \mathsf{ct})$ for these i, ct, $\mathcal{A}$ queries $\mathsf{G}(R)$. Note that this also includes indirect queries made by oracles $\textsc{Dec}, \textsc{Check}$.

In the code, we model this using a list $\mathcal{L}_R$ that contains the tuples (i, ct, R). The tuples are added in the query $\textsc{Enc}(i)$ and removed in queries $\textsc{Corr}(i)$ or $\textsc{Reveal}(i, \mathsf{ct})$. In each random oracle query $\mathsf{G}(R)$, we check if such a tuple is currently in the list and set $\mathsf{badR} := 1$ in this case.

As the ciphertext ct still contains information about R (namely, in C_0, C_1), we do not bound the probability of badR, but instead delay its analysis. We have

$$|\mathsf{Adv}_0 - \mathsf{Adv}_1| \leq \Pr[\mathsf{badR} \text{ in } \mathbf{G}_1].$$

Game $\mathbf{G}_2$: In game $\mathbf{G}_2$, we change how the matrix $\mathbf{A}$ is generated. Recall that before, it is generated as $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$. In game $\mathbf{G}_2$, we instead generate it as $\mathbf{A} := \bar{\mathbf{A}}\mathbf{C} + \mathbf{F}$ for $\bar{\mathbf{A}} \xleftarrow{\$} \mathbb{Z}_q^{n \times k}$, $\mathbf{C} \xleftarrow{\$} \mathbb{Z}_q^{k \times m}$, and $\mathbf{F} \leftarrow D_{\mathbb{Z},\gamma}^{n \times m}$. It is easy to see that we can bound the difference between $\mathbf{G}_1$ and $\mathbf{G}_2$, by n applications (one for each row of $\mathbf{A}$) of $\mathsf{LWE}_{k,m,q,D_{\mathbb{Z},\gamma}}$, or equivalently, an n-fold LWE assumption. The corresponding reduction $\mathcal{B}$ gets $\mathbf{A}, \mathbf{C}$ as input, and uses $\mathsf{par} := \mathbf{A}$. Then, it simulates $\mathbf{G}_1$, and outputs whatever the

game outputs. It follows that

$$|\mathsf{Adv}_1 - \mathsf{Adv}_2| \leq n \cdot \mathsf{Adv}_{\mathcal{B}}^{\mathsf{LWE}_{k,m,q,D_{\mathbb{Z},\gamma}}}(\lambda).$$

We can also give a similar reduction, that only outputs 1 if and only if event badR occurs, which can be checked efficiently. This implies that

$$|\Pr[\mathsf{badR} \text{ in } \mathbf{G}_1] - \Pr[\mathsf{badR} \text{ in } \mathbf{G}_2]| \leq n \cdot \mathsf{Adv}_{\mathcal{B}}^{\mathsf{LWE}_{k,m,q,D_{\mathbb{Z},\gamma}}}(\lambda).$$

Game $\mathbf{G}_3$: In game $\mathbf{G}_3$, we introduce another bad event $\mathsf{badK} := \mathsf{badK0} \vee \mathsf{badK1}$, and let the game abort if it occurs. This event is very similar to event badR. Namely, we consider a query of the form $\textsc{Enc}(i)$. Recall that in this query, values $\mathbf{x}, \hat{\mathbf{h}}_0, \mathbf{h}_0$ and $\hat{\mathbf{h}}_1, \mathbf{h}_1$ are defined. Then, the oracle sets $\hat{K}_0 := \mathsf{H}(\mathbf{x}, \hat{\mathbf{h}}_0, \mathbf{h}_0)$ and $\hat{K}_1 := \mathsf{H}(\mathbf{x}, \hat{\mathbf{h}}_1, \mathbf{h}_1)$. Later, a ciphertext ct is returned to $\mathcal{A}$. We say that the event $\mathsf{badK0}$ occurs, if one of the following holds:

- At this point, $\mathcal{A}$ already queried $\mathsf{H}(\mathbf{x}, \hat{\mathbf{h}}_{1-b_i}, \mathbf{h}_{1-b_i})$, or
- at some later point, before any query of the form $\textsc{Corr}(i)$ or $\textsc{Reveal}(i, \mathsf{ct})$ for these i, ct, $\mathcal{A}$ queries $\mathsf{H}(\mathbf{x}, \hat{\mathbf{h}}_{1-b_i}, \mathbf{h}_{1-b_i})$.

Similarly, we say that the event $\mathsf{badK1}$ occurs, if one of the following holds:

- At this point, $\mathcal{A}$ already queried $\mathsf{H}(\mathbf{x}, \hat{\mathbf{h}}_{b_i}, \mathbf{h}_{b_i})$, or
- at some later point, before any query of the form $\textsc{Corr}(i)$ or $\textsc{Reveal}(i, \mathsf{ct})$ for these i, ct, $\mathcal{A}$ queries $\mathsf{H}(\mathbf{x}, \hat{\mathbf{h}}_{b_i}, \mathbf{h}_{b_i})$.

As for event badR, this also includes indirect queries made by oracles $\textsc{Dec}, \textsc{Check}$. Similar to event badR, we formally model these two events via lists $\mathcal{L}_{K,0}$ and $\mathcal{L}_{K,1}$, where $\mathcal{L}_{K,0}$ is associated to event $\mathsf{badK0}$ and $\mathcal{L}_{K,1}$ is associated to event $\mathsf{badK1}$. Also, note that although we defined the events with respect to bit b_i, their symmetry ensures that there is no additional information about b_i given to $\mathcal{A}$. Clearly, we have

$$|\Pr[\mathsf{badR} \text{ in } \mathbf{G}_2] - \Pr[\mathsf{badR} \text{ in } \mathbf{G}_3]| \leq \Pr[\mathsf{badK} \text{ in } \mathbf{G}_3]$$

and

$$\begin{aligned} |\mathsf{Adv}_2 - \mathsf{Adv}_3| &\leq \Pr[\mathsf{badK} \text{ in } \mathbf{G}_3] \\ &\leq \Pr[\mathsf{badK0} \text{ in } \mathbf{G}_3] + \Pr[\mathsf{badK1} \text{ in } \mathbf{G}_3]. \end{aligned}$$

We upper bound these probabilities in Lemma 3.5 and Lemma 3.6. Also, we upper bound the probability of badR in $\mathbf{G}_3$ in Lemma 3.4.

Further, we claim that the probability that $\mathbf{G}_3$ outputs 1 is negligible. To see this, consider the final output $(i^*, \mathsf{ct}^*, K^*)$ of $\mathcal{A}$. If the game outputs 1, then ct^* has been output by $\textsc{Enc}(i^*)$, and $K^* = R$, where R has been sampled in the query $\textsc{Enc}(i^*)$ that returned ct^*. Further, if the game outputs 1, then the events badR and badK did not occur. The game can only output if $\mathcal{A}$ never queried $\textsc{Corr}(i^*)$, and never queried $\textsc{Reveal}(i^*, \mathsf{ct}^*)$, and therefore we know that $\mathcal{A}$ never queried $\mathsf{G}(R)$ and never queried $\mathsf{H}(\mathbf{x}, \hat{\mathbf{h}}_{b_i}, \mathbf{h}_{b_i})$ or $\mathsf{H}(\mathbf{x}, \hat{\mathbf{h}}_{1-b_i}, \mathbf{h}_{1-b_i})$, where the values $\mathbf{x}, \hat{\mathbf{h}}_0, \hat{\mathbf{h}}_1, \mathbf{h}_0, \mathbf{h}_1$ have been defined in the query $\textsc{Enc}(i^*)$ that returned ct^*. This means that the ciphertext ct^* information-theoretically hides the value R, and R is distributed uniformly at random from $\mathcal{A}$'s point of view. Thus, the probability that $K^* = R$ is at most $2^{-\lambda}$, and we get

$$\mathsf{Adv}_3 \leq \frac{Q_{\mathsf{E}}}{2^{\lambda}}.$$

In summary, we can upper bound Adv_0 by

$$\begin{aligned} & \Pr[\mathsf{badR} \text{ in } \mathbf{G}_1] + n \cdot \mathsf{Adv}_{\mathcal{B}}^{\mathsf{LWE}_{k,m,q,D_{\mathbb{Z},\gamma}}}(\lambda) + \Pr[\mathsf{badK} \text{ in } \mathbf{G}_3] + \frac{Q_{\mathsf{E}}}{2^{\lambda}} \\ \leq\ & \Pr[\mathsf{badR} \text{ in } \mathbf{G}_3] + 2n \cdot \mathsf{Adv}_{\mathcal{B}}^{\mathsf{LWE}_{k,m,q,D_{\mathbb{Z},\gamma}}}(\lambda) + 2 \cdot \Pr[\mathsf{badK1} \text{ in } \mathbf{G}_3] + \mathsf{negl}(\lambda) \\ \leq\ & 2n \cdot \mathsf{Adv}_{\mathcal{B}}^{\mathsf{LWE}_{k,m,q,D_{\mathbb{Z},\gamma}}}(\lambda) + 2 \cdot \Pr[\mathsf{badK1} \text{ in } \mathbf{G}_3] + \mathsf{negl}(\lambda) \\ \leq\ & 2n \cdot \mathsf{Adv}_{\mathcal{B}}^{\mathsf{LWE}_{k,m,q,D_{\mathbb{Z},\gamma}}}(\lambda) + 2\left(2n \cdot \mathsf{Adv}_{\mathcal{B}}^{\mathsf{LWE}_{k,m,q,D_{\mathbb{Z},\gamma}}}(\lambda) + \mathsf{negl}(\lambda)\right) + \mathsf{negl}(\lambda) \\ \leq\ & 6n \cdot \mathsf{Adv}_{\mathcal{B}}^{\mathsf{LWE}_{k,m,q,D_{\mathbb{Z},\gamma}}}(\lambda) + \mathsf{negl}(\lambda). \end{aligned}$$

□

Lemma 3.4 *With assumptions from the proof of Theorem 3.3, we have*

$$\Pr\left[\mathsf{badR} \text{ in } \boldsymbol{G}_3\right] \leq \frac{2Q_{\mathsf{E}}Q_{\mathsf{G}}}{2^{\lambda}} \leq \mathsf{negl}(\lambda).$$

The proof of the lemma is postponed to Appendix B.

Lemma 3.5 *With assumptions from the proof of Theorem 3.3, we have*

$$\Pr\left[\mathsf{badK0} \text{ in } \boldsymbol{G}_3\right] \leq \mathsf{negl}(\lambda).$$

Proof. First, we write the event $\mathsf{badK0}$ as

$$\mathsf{badK0} = \bigvee_{j \in [Q_{\mathsf{E}}]} \mathsf{badK0}_j,$$

where Q_{E} is the number of $\mathcal{A}$'s queries to Enc and $\mathsf{badK0}_j$ denotes the event that $\mathsf{badK0}_j$ occurs for the entry $(\mathsf{qry}, i, \mathsf{ct})$ that is inserted into list $\mathcal{L}_{K,0}$ in the jth query to oracle Enc. We bound the probability of each $\mathsf{badK0}_j$ separately, and conclude with a union bound.

To this end, fix $j \in [Q_{\mathsf{E}}]$. We define a new game $\mathbf{G}'_j$, which is defined to be as game $\mathbf{G}_3$, but with the following change: Consider the jth query to oracle Enc. Assume that in this query an entry $(\mathsf{qry}, i, \mathsf{ct})$ is inserted into list $\mathcal{L}_{K,0}$. Game $\mathbf{G}'_j$ immediately outputs 1 as soon as $\mathsf{badK0}_j$ occurs, i.e. if either $\mathsf{H}(\mathsf{qry})$ is already defined before the query, or $\mathsf{H}(\mathsf{qry})$ is queried before queries $\textsc{Corr}(i)$ or $\textsc{Reveal}(i, \mathsf{ct})$. Also, if $\mathsf{badK0}_j$ can no longer occur (i.e. $\mathcal{A}$ queries $\textsc{Corr}(i)$ or $\textsc{Reveal}(i, \mathsf{ct})$), the game immediately outputs 0. Note that until game $\mathbf{G}'_j$ outputs something, the view of $\mathcal{A}$ is identical to its view in $\mathbf{G}_3$. This implies that

$$\Pr\left[\mathsf{badK0}_j \text{ in } \mathbf{G}_3\right] = \Pr\left[\mathbf{G}'_j \Rightarrow 1\right].$$

Next, we change $\mathbf{G}'_j$ into $\mathbf{G}''_j$. In this game, consider the jth query to oracle Enc again. Recall that in this oracle query, a vector $\hat{\mathbf{h}}_{1-b_i}$ is defined via

$$\hat{\mathbf{h}}_{1-b_i} := \mathbf{U}^t_{i,1-b_i}\mathbf{s} + \mathbf{h}_{1-b_i}\lfloor q/2 \rceil.$$

In this game, we instead sample $\hat{\mathbf{h}}_{1-b_i} \xleftarrow{\$} \mathbb{Z}_q^{\lambda}$. We argue that the games are statistically close by using the generalized leftover hash lemma (Lemma 2.5), where the hash function is given by $\mathbf{s} \mapsto \mathbf{U}^t_{i,1-b_i}\mathbf{s}$. Note that $\mathbf{U}_{i,1-b_i}$ is sampled uniformly at random in $\mathbf{G}'_j$, and therefore this constitutes a universal family of hash functions. To use the generalized leftover hash lemma, we first need to lower bound the entropy of $\mathbf{s}$. To this end, we make use of Lemma 2.4.

Observe that in the jth query to oracle Enc, the only information (apart from $\mathbf{U}^t_{i,1-b_i}\mathbf{s}$) that $\mathcal{A}$ gets is $\mathbf{x}$ and $\hat{\mathbf{h}}_{b_i}$. Let $\epsilon = 2^{-\lambda}$. Then, we can use Lemma 2.4 (note that $\mathbf{A}$ has the correct form $\bar{\mathbf{A}}\mathbf{C} + \mathbf{F}$) to get

$$\begin{aligned}
\tilde{H}^{\epsilon}_{\infty}\left(\mathbf{s}\middle|\mathbf{x}, \hat{\mathbf{h}}_{b_i}\right) &\geq \tilde{H}^{\epsilon}_{\infty}\left(\mathbf{s}|\mathbf{x}\right) - \lambda \log q \\
&= \tilde{H}^{\epsilon}_{\infty}\left(\mathbf{s}\middle|\mathbf{A}^t\mathbf{s} + \mathbf{e}\right) - \lambda \log q \geq H_{\infty}\left(\mathbf{s}\right) - (k + 3\lambda)\log q - \mathsf{negl}(\lambda) \\
&= n\log(2\eta + 1) - (k + 3\lambda)\log q - \mathsf{negl}(\lambda) \geq \lambda \log q + \Omega(n).
\end{aligned}$$

For the last inequality, we used our assumption about the parameters. Now, we can use Lemma 2.5 with $\epsilon = 2^{-\lambda}$ and $\mathcal{Y} := \mathbb{Z}_q^{\lambda}$, and get that the statistical distance is at most

$$2\epsilon + \frac{1}{2}\sqrt{2^{-\tilde{H}^{\epsilon}_{\infty}\left(\mathbf{s}|\mathbf{x}, \hat{\mathbf{h}}_{b_i}\right)} \cdot |\mathcal{Y}|} \leq 2^{-\lambda+1} + \frac{1}{2}\sqrt{2^{-\lambda\log q - \Omega(n) + \lambda \log q}} \leq \mathsf{negl}(\lambda).$$

Thus, we have

$$\left|\Pr\left[\mathbf{G}'_j \Rightarrow 1\right] - \Pr\left[\mathbf{G}''_j \Rightarrow 1\right]\right| \leq \mathsf{negl}(\lambda).$$

Remark. The subtlety in the above argument is that oracles DEC and CHECK depend on the secret key. As the secret key is statistically independent of the value $\mathbf{s}$, the argument goes through even if the adversary had the secret key. However, this only holds under the assumption that event badR does not occur, as otherwise decrypting and reencrypting (and therefore querying $\mathsf{G}(R)$) would reveal $\mathbf{s}$.

Finally, we bound the probability that $\mathbf{G}''_j$ outputs 1. Recall that in the jth query to oracle ENC, the value $\mathbf{h}_{1-b_i}$ is part of the output of $\mathsf{G}(R)$. As we assume that badR does not occur, this means that during the query, this is a fresh value sampled uniformly at random. Also, our previous change removed all information about $\mathbf{h}_{1-b_i}$ from the response of oracle ENC. Thus, this value is uniformly random from $\mathcal{A}$'s view for the entire game. As it is part of the random oracle query qry that triggers event $\mathsf{badK0}_j$ and lets the game output 1, we can use a union bound over all random oracle queries and get

$$\Pr\left[\mathbf{G}''_j \Rightarrow 1\right] \leq \frac{Q_{\mathsf{H}}}{2^\lambda} \leq \mathsf{negl}(\lambda).$$

□

Lemma 3.6 *With assumptions from the proof of Theorem 3.3, there is a PPT algorithm $\mathcal{B}$ with $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A})$ and*

$$\Pr\left[\mathsf{badK1} \text{ in } \boldsymbol{G}_3\right] \leq 2n \cdot \mathsf{Adv}_{\mathcal{B}}^{\mathsf{LWE}_{k,m,q,D_{\mathbb{Z},\gamma}}}(\lambda) + \mathsf{negl}(\lambda).$$

Proof. To bound the probability, we introduce games $\mathbf{G}_4, \mathbf{G}_5$ and $\mathbf{G}_6$. We argue that the probability of $\mathsf{badK1}$ does not change significantly from $\mathbf{G}_3$ to $\mathbf{G}_6$. In game $\mathbf{G}_6$, we use Lemma 3.7 to argue that the view of $\mathcal{A}$ is independent of the bit b_i as long as the events $\mathsf{badK0}$ and $\mathsf{badK1}$ can occur. This will imply that the probabilities of $\mathsf{badK0}$ and $\mathsf{badK1}$ are the same in $\mathbf{G}_6$. Therefore, we can just bound the probability of $\mathsf{badK0}$ in $\mathbf{G}_6$. This can easily be done by going back to game $\mathbf{G}_3$ and using Lemma 3.5. We will now proceed in more detail.

Game $\mathbf{G}_4$: Game $\mathbf{G}_4$ is as $\mathbf{G}_3$, but we change how $\mathsf{par} := \mathbf{A}$ is sampled. Concretely, we revert the change from $\mathbf{G}_1$ to $\mathbf{G}_2$ and sample $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$. As from $\mathbf{G}_1$ to $\mathbf{G}_2$, it follows that

$$|\Pr\left[\mathsf{badK1} \text{ in } \mathbf{G}_3,\right] - \Pr\left[\mathsf{badK1} \text{ in } \mathbf{G}_4\right]| \leq n \cdot \mathsf{Adv}_{\mathcal{B}}^{\mathsf{LWE}_{k,m,q,D_{\mathbb{Z},\gamma}}}(\lambda),$$
$$|\Pr\left[\mathsf{badK0} \text{ in } \mathbf{G}_3\right] - \Pr\left[\mathsf{badK0} \text{ in } \mathbf{G}_4\right]| \leq n \cdot \mathsf{Adv}_{\mathcal{B}}^{\mathsf{LWE}_{k,m,q,D_{\mathbb{Z},\gamma}}}(\lambda).$$

Game $\mathbf{G}_5$: Game $\mathbf{G}_5$ is as $\mathbf{G}_4$, but we change how the matrices $\mathbf{U}_{i,1-b_i}$ are defined for all $i \in [N]$. Recall that before, these are sampled uniformly at random from $\mathbb{Z}_q^{n \times \lambda}$. In $\mathbf{G}_5$, we first sample $\mathbf{Z}_{i,1-b_i} \leftarrow D_{\mathbb{Z},\alpha}^{m \times \lambda}$ and then set $\mathbf{U}_{i,1-b_i} := \mathbf{A}\mathbf{Z}_{i,1-b_i}$. By Lemma 2.2, the distributions of these matrices are statistically close to uniform. Thus we get

$$|\Pr\left[\mathsf{badK1} \text{ in } \mathbf{G}_4\right] - \Pr\left[\mathsf{badK1} \text{ in } \mathbf{G}_5\right]| \leq \mathsf{negl}(\lambda),$$
$$|\Pr\left[\mathsf{badK0} \text{ in } \mathbf{G}_4\right] - \Pr\left[\mathsf{badK0} \text{ in } \mathbf{G}_5\right]| \leq \mathsf{negl}(\lambda).$$

Game $\mathbf{G}_6$: Game $\mathbf{G}_6$ is as $\mathbf{G}_5$, but we add additional bad events, that let the game abort. Namely, the game aborts as soon as one of the following occurs:

- For some $i \in [N]$, one of the columns $\mathbf{z}$ of $\mathbf{Z}_{i,0}$ or $\mathbf{Z}_{i,1}$ satisfies $\|\mathbf{z}\| > \alpha\sqrt{m}$.
- For some random oracle query $\mathsf{G}(R)$ that returns $(\mathbf{s}, \rho, \mathbf{h}_0, \mathbf{h}_1)$, it holds that $\|\mathsf{SampleD}(m, \alpha'; \rho)\| > \alpha'\sqrt{m}$.
- There is a collision in random oracle H.

By Lemma 2.3, the first two events occur only with negligible probability. As hash values for H are sampled uniformly at random from $\{0,1\}^\lambda$, the third event also occurs only with negligible probability. It follows that

$$|\Pr\left[\mathsf{badK1} \text{ in } \mathbf{G}_5\right] - \Pr\left[\mathsf{badK1} \text{ in } \mathbf{G}_6\right]| \leq \mathsf{negl}(\lambda),$$
$$|\Pr\left[\mathsf{badK0} \text{ in } \mathbf{G}_5\right] - \Pr\left[\mathsf{badK0} \text{ in } \mathbf{G}_6\right]| \leq \mathsf{negl}(\lambda).$$

Finally, we claim that

$$\Pr\left[\mathsf{badK1} \text{ in } \mathbf{G}_6\right] = \Pr\left[\mathsf{badK0} \text{ in } \mathbf{G}_6\right].$$

Note that once we showed this, the statement follows.

It remains to show the claim. First, fix a point in the execution of the game, and let $\mathcal{I} \subseteq [N]$ denote the set of indices $i \in [N]$, for which $\mathcal{A}$ did not yet query $\textsc{Corr}(i)$ at this point. Note that the events $\mathsf{badK0}$ and $\mathsf{K1}$ can only occur for indices in $\mathcal{I}$. We claim that the view of $\mathcal{A}$ at this point does not depend on the bits b_i for $i \in \mathcal{I}$. This is because the public keys itself do not reveal b_i (because both $\mathbf{U}_{i,0}$ and $\mathbf{U}_{i,1}$ have the same distribution), the queries to $\textsc{Enc}$ and $\textsc{Reveal}$ do not reveal b_i, and, by Lemma 3.7, queries to $\textsc{Dec}$ and $\textsc{Check}$ do not reveal b_i. Second, note that the events $\mathsf{badK0}, \mathsf{badK1}$ are exactly the same, except for the lists $\mathcal{L}_{K,0}$ and $\mathcal{L}_{K,1}$ that appear in their definition. Also, the lists only differ depending on the bit b_i. As $\mathcal{A}$ has no information about b_i, and the events $\mathsf{badK0}$ and $\mathsf{badK1}$ occur with equal probability. □

Lemma 3.7 *Let $4\alpha\alpha' m < q$. Let $\mathbf{Z}_0, \mathbf{Z}_1 \in \mathbb{Z}_q^{m\times\lambda}$ such that $\|\mathbf{z}_{b,i}\| \leq \alpha\sqrt{m}$ for all $i \in [\lambda], b \in \{0,1\}$, where $\mathbf{z}_{b,i}$ denotes column i of $\mathbf{Z}_b$. Let $\mathsf{par} := \mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and $\mathbf{U}_0 := \mathbf{A}\mathbf{Z}_0$, $\mathbf{U}_1 := \mathbf{A}\mathbf{Z}_1$. Assume that for all outputs $(\mathbf{s}, \rho, \mathbf{h}_0, \mathbf{h}_1)$ of G and $\mathbf{e} \leftarrow \mathsf{SampleD}(m, \alpha'; \rho)$ it holds that $\|\mathbf{e}\| \leq \alpha'\sqrt{m}$. Further, assume that for each $x \neq x' \in \{0,1\}^*$ we have $\mathsf{H}(x) \neq \mathsf{H}(x')$. Then, for each $\mathsf{ct} = (C_0, C_1, \mathbf{x}, \hat{\mathbf{h}}_0, \hat{\mathbf{h}}_1) \in \{0,1\}^\lambda \times \{0,1\}^\lambda \times \mathbb{Z}_q^m \times \mathbb{Z}_q^\lambda \times \mathbb{Z}_q^\lambda$ it holds that $\mathsf{Decap}((\mathbf{Z}_0, 0), \mathsf{ct}) = \mathsf{Decap}((\mathbf{Z}_1, 1), \mathsf{ct})$.*

The proof of the lemma is postponed to Appendix B.

3.3 Construction from Matrix Decisional Diffie-Hellman

We construct an $\mathsf{OW\text{-}ChCCA}$ secure key encapsulation mechanism based on the (matrix) decisional Diffie-Hellman assumption [EHK+13], which has deterministic ciphertext derivation. The construction mimics our lattice-based construction. We postpone the formal description and details to Appendix C.

4 AKE from OW-ChCCA Secure KEMs

A two-message AKE protocol $\mathsf{AKE} := (\mathsf{Setup}', \mathsf{KG}', \mathsf{Init}, \mathsf{Der}_\mathsf{R}, \mathsf{Der}_\mathsf{I})$ consists of five algorithms. The setup algorithm Setup', on input security parameter 1^λ, outputs global AKE system parameters par'. KG' takes the system parameters par' as input and outputs a long-term key pair $(\mathsf{pk}', \mathsf{sk}')$ for one party.

Let P_i and P_j as two parties and $(\mathsf{pk}'_i, \mathsf{sk}'_i)$ and $(\mathsf{pk}'_j, \mathsf{sk}'_j)$ be the long-term key pair of P_i and P_j, respectively. Figure 7 shows how P_i, (as initiator) establish a shared key with P_j (as responder). To initialize the session with P_j, P_i runs the initialization algorithm Init, which takes $\mathsf{sk}'_i, \mathsf{pk}'_j$ as inputs and outputs a protocol message M_i and session state st, and then P_i sends M_i to P_j. On receiving M_i, P_j runs the responder's derivation algorithm Der_R, which takes sk'_j, pk'_i, and the received message M_i as input, to generate a responded message M_j and a session key SK_j. P_j sends M_j to P_i. Finally, on receiving M_j, P_i runs the initiator's derivation algorithm Der_I which inputs $\mathsf{sk}'_i, \mathsf{pk}'_j$, the received message M_j, and the session state st generated before, to generate a session key sk'_i. In two-message AKE protocols, the responder does not need to save session state since it can compute the session key right after receiving the initiator's message.

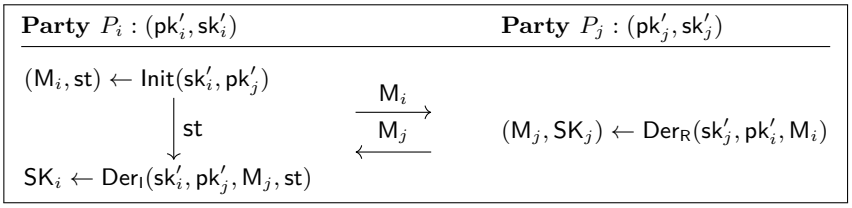

Figure 7: Illustration for a two-pass AKE protocol execution between party P_i and P_j.

We define the correctness of AKE protocols, stating that an honestly execution between two parties P_i and P_j as in Figure 7 will produce the same session key $\mathsf{SK}_i = \mathsf{SK}_j$.

Definition 4.1 (AKE Correctness). Let $\mathsf{AKE} := (\mathsf{Setup}', \mathsf{KG}', \mathsf{Init}, \mathsf{Der}_\mathsf{R}, \mathsf{Der}_\mathsf{I})$ be a AKE protocol. We say AKE is ρ-correct, if for any AKE system parameter $\mathsf{par}' \leftarrow \mathsf{Setup}'(1^\lambda)$, any $(\mathsf{pk}'_i, \mathsf{sk}'_i) \leftarrow$

Alg $\mathsf{Setup}'(1^\lambda)$
01 $\mathsf{par} \leftarrow \mathsf{Setup}_1(1^\lambda)$
02 $\widetilde{\mathsf{par}} \leftarrow \mathsf{Setup}_0(1^\lambda)$
03 **return** $\mathsf{par}' := (\widetilde{\mathsf{par}}, \mathsf{par})$

Alg $\mathsf{KG}'((\widetilde{\mathsf{par}}, \mathsf{par}))$
04 $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}_1(\mathsf{par})$
05 $\mathsf{k} \xleftarrow{\$} \{0,1\}^\kappa$
06 $(\mathsf{pk}', \mathsf{sk}') := (\mathsf{pk}, (\mathsf{sk}, \mathsf{k}))$
07 **return** $(\mathsf{pk}', \mathsf{sk}')$

Alg $\mathsf{Der}_\mathsf{R}((\mathsf{sk}_j, \mathsf{k}_j), \mathsf{pk}'_i, (\widetilde{\mathsf{pk}}, \mathsf{ct}_j))$
08 $\mathsf{K}_j := \mathsf{Decaps}_1(\mathsf{sk}'_j, \mathsf{ct}_j)$
09 **if** $\mathsf{K}_j = \perp$: **return** $\perp$
10 $(\widetilde{\mathsf{ct}}, \widetilde{\mathsf{K}}) \leftarrow \mathsf{Encaps}_0(\widetilde{\mathsf{pk}})$
11 $(\mathsf{ct}_i, \mathsf{K}_i) \leftarrow \mathsf{Encaps}_1(\mathsf{pk}_i)$
12 $\mathsf{ctxt} := (\mathsf{pk}'_i, \mathsf{pk}'_j, \widetilde{\mathsf{pk}}, \mathsf{ct}_i, \mathsf{ct}_j, \widetilde{\mathsf{ct}})$
13 $\mathsf{SK} := \mathsf{H}(\mathsf{ctxt}, \mathsf{K}_i, \mathsf{K}_j, \widetilde{\mathsf{K}})$
14 **return** $((\widetilde{\mathsf{ct}}, \mathsf{ct}_i), \mathsf{SK})$

Alg $\mathsf{Init}((\mathsf{sk}_i, \mathsf{k}_i), \mathsf{pk}'_j, (\widetilde{\mathsf{par}}, \mathsf{par}))$
15 $(\mathsf{ct}_j, \mathsf{K}_j) \leftarrow \mathsf{Encaps}_1(\mathsf{pk}'_j)$
16 $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}) \leftarrow \mathsf{KG}_0(\widetilde{\mathsf{par}})$
17 $\mathsf{st}' := (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}, \mathsf{ct}_j, \mathsf{K}_j)$
18 $IV \xleftarrow{\$} \{0,1\}^\kappa$
19 $\mathsf{st} := (IV, \mathsf{G}(\mathsf{k}_i, IV) \oplus \mathsf{st}')$
20 **return** $((\widetilde{\mathsf{pk}}, \mathsf{ct}_j), \mathsf{st})$

Alg $\mathsf{Der}_\mathsf{I}((\mathsf{sk}_i, \mathsf{k}_i), \mathsf{pk}'_j, (\widetilde{\mathsf{ct}}, \mathsf{ct}_i), \mathsf{st})$
21 **let** $(IV, \varphi) := \mathsf{st}$
22 $\mathsf{st}' := \mathsf{G}(IV, \mathsf{k}_i) \oplus \varphi$
23 **let** $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}, \mathsf{ct}_j, \mathsf{K}_j) := \mathsf{st}'$
24 $\mathsf{K}_i := \mathsf{Decaps}_1(\mathsf{sk}'_i, \mathsf{ct}_i)$
25 $\widetilde{\mathsf{K}} := \mathsf{Decaps}_0(\widetilde{\mathsf{sk}}, \widetilde{\mathsf{ct}})$
26 **if** $\mathsf{K}_i = \perp \vee \widetilde{\mathsf{K}} = \perp$: **return** $\perp$
27 $\mathsf{ctxt} := (\mathsf{pk}'_i, \mathsf{pk}'_j, \widetilde{\mathsf{pk}}, \mathsf{ct}_i, \mathsf{ct}_j, \widetilde{\mathsf{ct}})$
28 $\mathsf{SK} := \mathsf{H}(\mathsf{ctxt}, \mathsf{K}_i, \mathsf{K}_j, \widetilde{\mathsf{K}})$
29 **return** SK

Figure 8: Our direct AKE protocol AKE. Lines with purple are used to achieve security against internal states reveal by encrypting state st'. The only difference in our indirect AKE protocol $\mathsf{AKE}_{\mathsf{in}}$ is that session keys in Lines 13 and 28 are computed as $\mathsf{SK} := \mathsf{H}(\mathsf{ctxt}, \mathsf{H}'(\mathsf{pk}_i, \mathsf{ct}_i, \mathsf{K}_i), \mathsf{H}'(\mathsf{pk}_j, \mathsf{ct}_j, \mathsf{K}_j), \mathsf{H}'(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{ct}}, \widetilde{\mathsf{K}}))$ where H' is a different hash function.

$\mathsf{KG}'(\mathsf{par}'), (\mathsf{pk}'_j, \mathsf{sk}'_j) \leftarrow \mathsf{KG}'(\mathsf{par}')$, the following probability is at least ρ.

$$\Pr\left[\mathsf{SK}_j = \mathsf{SK}_i \;\middle|\; \begin{array}{c}(\mathsf{M}_i, \mathsf{st}) \leftarrow \mathsf{Init}(\mathsf{sk}'_i, \mathsf{pk}'_j) \\ (\mathsf{M}_j, \mathsf{SK}_j) \leftarrow \mathsf{Der}_\mathsf{R}(\mathsf{sk}'_j, \mathsf{pk}'_i, \mathsf{M}_i) \\ \mathsf{SK}_i \leftarrow \mathsf{Der}_\mathsf{I}(\mathsf{sk}'_i, \mathsf{pk}'_j, \mathsf{M}_j, \mathsf{st})\end{array}\right]$$

AKE Security Model. Following [JKRS21], we define a game-based AKE security model using pseudocode. We use the weak-forward-secrecy model wFS-St in [JKRS21] which captures some attacks against AKE such as key-compromise-impersonation (KCI) and maximal-exposure (MEX) and considers weak forward security. Details of wFS-St model are shown in Appendix D.

4.1 Our AKE Protocol

Let $\mathsf{KEM}_1 = (\mathsf{Setup}_1, \mathsf{KG}_1, \mathsf{Encaps}_1, \mathsf{Decaps}_1)$ and $\mathsf{KEM}_0 = (\mathsf{Setup}_0, \mathsf{KG}_0, \mathsf{Encaps}_0, \mathsf{Decaps}_0)$ be two KEM schemes. We construct our direct two-message AKE protocol $\mathsf{AKE} = (\mathsf{Setup}', \mathsf{KG}', \mathsf{Init}, \mathsf{Der}_\mathsf{R}, \mathsf{Der}_\mathsf{I})$ as shown in Figure 8, where $\mathsf{H} : \{0,1\}^* \to \mathcal{SK}$ is a hash function which is used to derive the session key, and $\mathsf{G} : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^d$ is a hash function which outputs a one-time key to encrypt state. We assume that any unencrypted state, $\mathsf{st}' = (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}, \mathsf{ct}_j, \mathsf{K}_j)$ as in Figure 8 can be encoded as a d-bit string.

In Section 5.1 we show that an OW-ChCCA secure KEM gives us a tight non-committing KEM. By the work of Jager et al.[JKRS21], this will give us a tightly secure AKE indirectly. This is our second approach in constructing tightly secure AKE, and it is indirect. The only difference between our direct protocol AKE and our indirect one $\mathsf{AKE}_{\mathsf{in}}$ is the session key derivation as described in the caption of Figure 8.

Correctness. The correctness of our AKE protocol is dependent on KEM_1 or KEM_0. Suppose that KEM_1 is $(1-\delta_1)$-correct and KEM_0 is $(1-\delta_0)$-correct (cf. Definition 2.1). In our protocol (Figure 8), each session includes two KEM_1 ciphertexts and one KEM_0 ciphertext. By the union bound, the probability that for honest matching sessions sID and sID', they do not produce the same key is at most $2\delta_1 + \delta_0$, so we have the following lemma.

Lemma 4.2 *If* KEM_1 *is* $(1-\delta_1)$*-correct and* KEM_0 *is* $(1-\delta_0)$*-correct, then the AKE protocol* AKE *in Figure 8 is* $(1-2\delta_1-\delta_0)$*-correct.*

Security. Theorem 4.3 shows that if KEM_1 and KEM_0 are OW-ChCCA, and G and H are modeled as random oracles, then AKE is tightly wFS-St secure. We postpone the proof of Theorem 4.3 to Appendix E.

Theorem 4.3 *Let* $\mathsf{H}:\{0,1\}^* \to \mathcal{SK}$ *and* $\mathsf{G}:\{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^d$ *be random oracles. If* KEM_1 *is* $(1-\delta_1)$*-correct for* $\delta_1 = \mathsf{negl}(\lambda)$ *and* OW-ChCCA *secure with* $\gamma_1 = \omega(\log(\lambda))$ *bits ciphertext entropy and* $\mu_1 = \omega(\log(\lambda))$ *bits public key entropy, and* KEM_0 *is* $(1-\delta_0)$*-correct for* $\delta_0 = \mathsf{negl}(\lambda)$ *and* OW-ChCCA *secure with* $\gamma_0 = \omega(\log(\lambda))$ *bits ciphertext entropy and* $\mu_0 = \omega(\log(\lambda))$ *bits public key entropy, then the AKE protocol* AKE *in Figure 8 is* wFS-St *secure.*

For any PPT adversary $\mathcal{A}$ *against* wFS-St *security of* AKE*, there are PPT algorithm* $\mathcal{B}_1$ *and* $\mathcal{B}_0$ *with* $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B}_1)$ *and* $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B}_0)$ *and*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{wFS\text{-}St}}_{\mathcal{A},\mathsf{AKE}}(\lambda) \le\ & 2\mathsf{Adv}^{\mathsf{OW\text{-}ChCCA}}_{\mathcal{B}_1,\mathsf{KEM}_1}(\lambda) + 2\mathsf{Adv}^{\mathsf{OW\text{-}ChCCA}}_{\mathcal{B}_0,\mathsf{KEM}_0}(\lambda) \\ & + 2\delta_1 + 2\delta_0 + \frac{(N+1)\cdot S\cdot Q_{\mathsf{G}}}{2^{\kappa-1}} + \frac{2(Q_{\mathsf{H}}^2+S^2)}{|\mathcal{SK}|} \\ & + 2S\cdot(Q_{\mathsf{H}}+S)\cdot\left(\frac{1}{2^{\gamma_1}}+\frac{1}{2^{\gamma_0}}+\frac{1}{2^{\mu_0}}\right) + \frac{Q_{\mathsf{G}}^2+N^2+S^2}{2^{d-1}},\end{aligned}$$

where Q_{G} *and* Q_{H} *are the numbers of queries to* G *and* H*, respectively.* N *and* S *are numbers of parties and sessions in the* wFS-St *security game, respectively.*

Perfect Forward Secrecy. Our protocol satisfies weak forward secrecy. To achieve tight (perfect) FS from LWE, one can combine our tightly secure OW-ChCCA KEM and multi-user SUF-CMA with corruptions. Such a signature scheme can be constructed from lattices by combining the Pan-Wagner signature scheme [PW22] and a chameleon hash function. Alternatively, we can combine our construction AKE with a MAC as key confirmation (the same as in HMQV [Kra05]) to get a more efficient construction. However, the resulting AKE is a three-message protocol, and its reduction to the MAC is the only non-tight part [GGJJ23]. However, when instantiating the MAC in the ROM, the loss only appears in statistical terms, and a security loss of symmetric primitives triggers significantly less efficiency penalty than for public key primitives (cf. Footnote 9 in [HJK+21]).

5 Further Applications of OW-ChCCA Security

We propose further applications of OW-ChCCA security. Namely, from an OW-ChCCA secure key encapsulation mechanism, we can tightly construct the following schemes:

- A non-committing key encapsulation mechanism, which implies a tightly secure AKE by [JKRS21]. The resulting AKE protocol, $\mathsf{AKE}_{\mathsf{in}}$, is described as in Figure 8;
- A public-key encryption scheme with simulation-based bi-selective opening security;
- A key encapsulation mechanism with enhanced CCA security [HLG21]. Since it is not the main result of our paper, we postpone this application to Appendix F.

5.1 Non-Committing Key Encapsulation Mechanism

Definition. We recall the definition of non-committing key encapsulation mechanisms (KEM) [JKRS21].

Definition 5.1 (N-Receiver Non-Committing KEM). Let $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encap}, \mathsf{Decap})$ be a key encapsulation mechanism which is relative to a simulator $\mathsf{Sim} = (\mathsf{SimGen}, \mathsf{SimEncaps}, \mathsf{SimHash})$. We define games $\mathsf{NC}_{\mathsf{real}}$ and $\mathsf{NC}_{\mathsf{sim}}$ as in Figure 9. The simulator Sim is only used in $\mathsf{NC}_{\mathsf{sim}}$. We say KEM is NC-CCA secure, if for all PPT adversaries $\mathcal{A}$, $\mathcal{A}$'s advantage is negligible:

$$\mathsf{Adv}^{\mathsf{NC\text{-}CCA}}_{\mathsf{KEM},\mathsf{Sim},\mathcal{A}}(\lambda) := \left|\Pr\left[\mathsf{NC}^{\mathcal{A}}_{\mathsf{real}}(\lambda) \Rightarrow 1\right] - \Pr\left[\mathsf{NC}^{\mathcal{A}}_{\mathsf{sim}}(\lambda) \Rightarrow 1\right]\right|$$

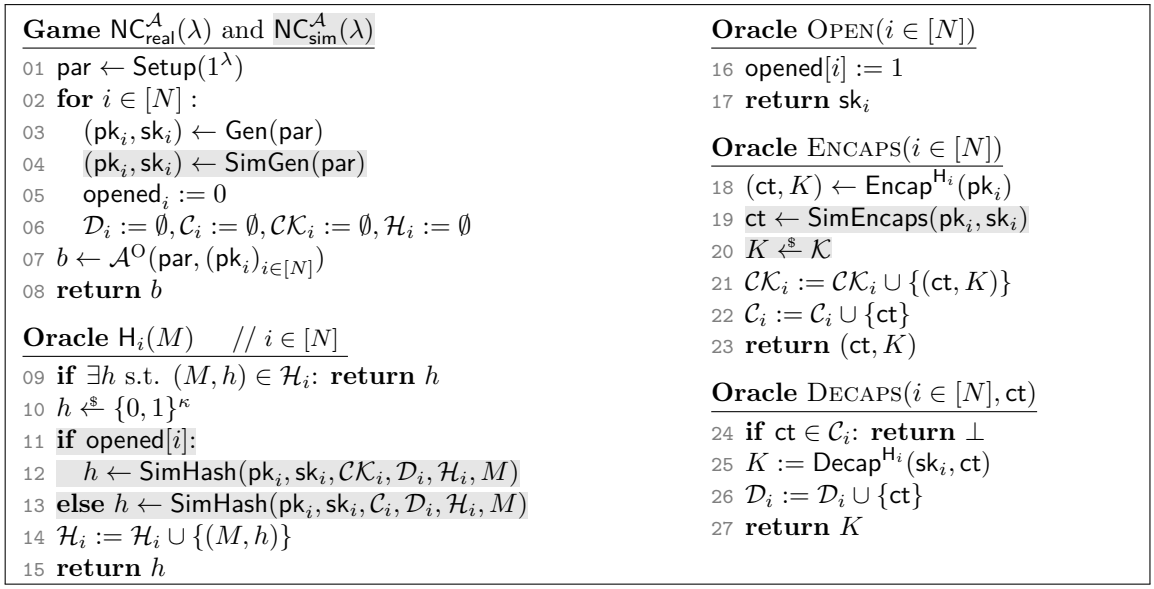

Figure 9: $\mathsf{NC}^{\mathcal{A}}_{\mathsf{real}}(\lambda)$ and $\mathsf{NC}^{\mathcal{A}}_{\mathsf{sim}}(\lambda)$ for an adversary $\mathcal{A}$ and $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encap}, \mathsf{Decap})$ with simulation processes $(\mathsf{SimGen}, \mathsf{SimEncaps}, \mathsf{SimHash})$. Algorithms Encap and Decap have oracle access to H, but not $\mathsf{SimEncaps}$. $\mathcal{A}$ has access to $\mathrm{O} := \{\mathsf{H}_1, ..., \mathsf{H}_N, \textsc{Encaps}, \textsc{Decaps}, \textsc{Open}\}$. Highlighted lines are only executed in $\mathsf{NC}^{\mathcal{A}}_{\mathsf{sim}}(\lambda)$.

Construction. In Figure 10, we transform a OW-ChCCA secure $\mathsf{KEM}_0 = (\mathsf{Setup}_0, \mathsf{Gen}_0, \mathsf{Encaps}_0, \mathsf{Decaps}_0)$ into a NC-CCA secure KEM scheme $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ using a random oracle $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^\kappa$, where $\kappa = \omega(\log(\lambda))$ is the key length of KEM. Following the definition in [JKRS21], Encaps and Decaps are associated with H. Namely, different users have access to different random oracles.

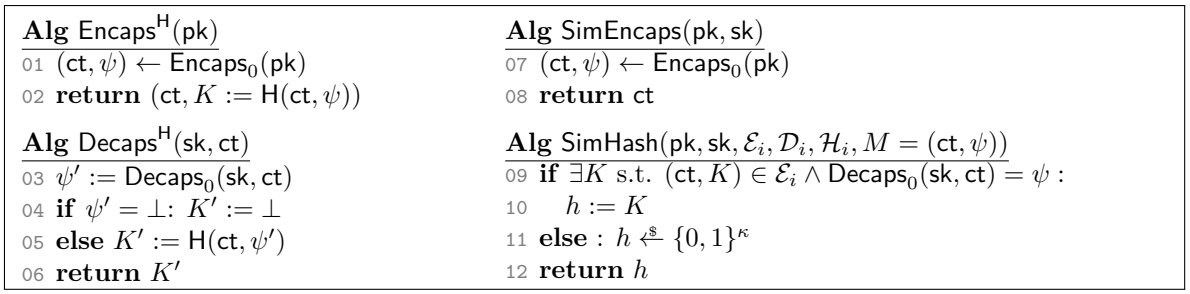

Figure 10: Key encapsulation mechanism $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ based on a KEM scheme $\mathsf{KEM}_0 = (\mathsf{Setup}_0, \mathsf{Gen}_0, \mathsf{Encaps}_0, \mathsf{Decaps}_0)$ and random oracle H, where $\mathsf{Setup} = \mathsf{Setup}_0$ and $\mathsf{Gen} = \mathsf{Gen}_0$. $\mathsf{Sim} = (\mathsf{SimGen}, \mathsf{SimEncaps}, \mathsf{SimHash})$ is a simulator of KEM, where $\mathsf{SimGen} = \mathsf{Gen}$. $\mathcal{E}_i$ is either $\mathcal{C}_i$ or $\mathcal{CK}_i$. $\mathcal{H}_i$ is a list that records the RO queries to H_i.

Theorem 5.2 *Let N be the number of users and let H_i for $i \in [N]$ be random oracles. If KEM_0 is* OW-ChCCA *secure, $(1-\delta_0)$-correct for $\delta_0 = \mathsf{negl}(\lambda)$, and has $\gamma_0 = \omega(\log(\lambda))$ bits of ciphertext entropy, then KEM is* NC-CCA *secure.*

Concretely, for any PPT algorithm $\mathcal{A}$, there is a PPT algorithm $\mathcal{B}$ such that $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B})$ and

$$\mathsf{Adv}^{\mathsf{NC\text{-}CCA}}_{\mathsf{KEM},\mathsf{Sim},\mathcal{A}}(\lambda) \leq 2 \cdot \mathsf{Adv}^{\mathsf{OW\text{-}ChCCA}}_{\mathcal{B},\mathsf{KEM}_0}(\lambda) + 3\delta_0 + \frac{Q_{\mathsf{H}}^2 + Q_{\mathsf{E}}^2}{2^{\kappa-1}} + \frac{NQ_{\mathsf{E}}(Q_{\mathsf{H}} + 2Q_{\mathsf{D}})}{2^{\gamma_0}},$$

where Q_{D}, Q_{E}, and Q_{H} are the numbers of $\mathcal{A}$'s queries to $\textsc{Decaps}, \textsc{Encaps}$, and $\{\mathsf{H}_i\}_{i\in[N]}$, respectively, and $\mathcal{B}$ also queries $\textsc{Enc}$ Q_{E} times.

Proof. Let $\mathcal{A}$ be an adversary against KEM in the $\mathsf{NC}^{\mathsf{KEM}}_{\mathsf{real}}(\lambda)$ game, where N is the number of users. Each user $i \in [N]$ is associated with public key pk_i. We prove Theorem 5.2 with games as defined in Figure 11 and Figure 12.

Game $\mathbf{G}_0$-$\mathbf{G}_3$

01 $\mathsf{par} \leftarrow \mathsf{Setup}_0(1^\lambda)$
02 **for** $i \in [N]$:
03 $\quad(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}_0(\mathsf{par})$
04 $\quad\mathsf{opened}[i] := 0$
05 $\quad\mathcal{D}_i := \emptyset, \mathcal{C}_i := \emptyset$
06 $\quad\mathcal{H}_i := \emptyset, \mathcal{CK}_i := \emptyset$
07 $\quad\mathcal{CK}^{\mathsf{ow}}_i := \emptyset$ // $\mathbf{G}_1$-$\mathbf{G}_2$
08 $b \leftarrow \mathcal{A}^{\mathrm{O}}(\mathsf{par}, (\mathsf{pk}_i)_{i\in[N]})$
09 **return** b

Oracle $\textsc{Open}(i \in [N])$

10 $\mathsf{opened}_i := 1$
11 **return** sk_i

Oracle $\textsc{Decaps}(i \in [N], \mathsf{ct})$

12 **if** $\mathsf{ct} \in \mathcal{C}_i$: **return** $\perp$
13 $\mathcal{D}_i := \mathcal{D}_i \cup \{\mathsf{ct}\}$
14 $\psi' := \mathsf{Decaps}_0(\mathsf{sk}_i, \mathsf{ct})$
15 **if** $\psi' = \perp$: $K' := \perp$
16 **else** : $K' := \mathsf{H}(\mathsf{ct}, \psi')$
17 **return** K'

Oracle $\textsc{Encaps}(i \in [N])$

18 $(\mathsf{ct}, \psi) \leftarrow \mathsf{Encaps}_0(\mathsf{pk}_i)$
19 $K := \mathsf{H}_i(\mathsf{ct}, \psi)$ // $\mathbf{G}_0$
20 $K \xleftarrow{\$} \{0,1\}^\kappa$ // $\mathbf{G}_1$-$\mathbf{G}_3$
21 $\mathcal{CK}^{\mathsf{ow}}_i := \mathcal{CK}^{\mathsf{ow}}_i \cup \{(\mathsf{ct}, \psi, K)\}$ // $\mathbf{G}_1$
22 $\mathcal{CK}_i := \mathcal{CK}_i \cup \{(\mathsf{ct}, K)\}$
23 $\mathcal{C}_i := \mathcal{C}_i \cup \{\mathsf{ct}\}$
24 **return** (ct, K)

Oracle $\mathsf{H}_i(\mathsf{ct}, \psi)$ // $i \in [N]$

25 **if** $\exists K$ s.t. $(\mathsf{ct}, \psi, K) \in \mathcal{H}_i$: **return** K
26 $K \xleftarrow{\$} \{0,1\}^\kappa$
27 **if** $\exists K'$ s.t. $(\mathsf{ct}, \psi, K') \in \mathcal{CK}^{\mathsf{ow}}_i$: // $\mathbf{G}_1$
28 $\quad K := K'$ // $\mathbf{G}_1$
29 **if** $\mathsf{ct} \in \mathcal{C}_i \wedge \neg\mathsf{opened}[i]$ // $\mathbf{G}_2$
30 $\quad\wedge\mathsf{Decaps}_0(\mathsf{sk}_i, \mathsf{ct}) = \psi$: // $\mathbf{G}_2$
31 $\quad$**abort** // $\mathbf{G}_2$
32 **if** $\mathsf{opened}[i]$: // $\mathbf{G}_2$-$\mathbf{G}_3$
33 $\quad$**if** $\exists K'$ s.t. $(\mathsf{ct}, K') \in \mathcal{CK}_i$ // $\mathbf{G}_2$-$\mathbf{G}_3$
34 $\quad\quad\wedge\mathsf{Decaps}_0(\mathsf{sk}_i, \mathsf{ct}) = \psi$: // $\mathbf{G}_2$-$\mathbf{G}_3$
35 $\quad\quad K := K'$ // $\mathbf{G}_2$-$\mathbf{G}_3$
36 $\mathcal{H}_i := \mathcal{H}_i \cup \{(\mathsf{ct}, \psi, K)\}$
37 **return** K

Figure 11: The games $\mathbf{G}_0$-$\mathbf{G}_3$ in the proof of Theorem 5.2. Lines with highlighted comments are only executed in the corresponding games.

Game $\mathbf{G}_0$: This game is the same as $\mathsf{NC}^{\mathsf{KEM}}_{\mathsf{real}}(\lambda)$, except that we exclude the collision among the outputs of H_i for all $i \in [N]$, and assume that challenge ciphertexts generated in $\textsc{Encaps}$ are never been issued to $\textsc{Decaps}$ before (i.e., if $\textsc{Encaps}(i)$ outputs a ciphertext ct, then $\mathsf{ct} \notin \mathcal{D}_i$). If such a collision happens at any time, then we abort the game. We do not explicitly define such events in the code for readability. By a union bound and the ciphertext entropy of KEM_0, we have

$$\left|\Pr\left[\mathsf{NC}^{\mathsf{KEM},\mathcal{A}}_{\mathsf{real}}(\lambda) \Rightarrow 1\right] - \Pr\left[\mathbf{G}^{\mathcal{A}}_0 \Rightarrow 1\right]\right| \leq \frac{Q^2_{\mathsf{H}} + Q^2_{\mathsf{E}}}{2^\kappa} + \frac{NQ_{\mathsf{E}}Q_{\mathsf{D}}}{2^{\gamma_0}}.$$

Game $\mathbf{G}_1$: We modify the generation of challenge keys and simulation of $\{\mathsf{H}_i\}_{i\in[N]}$. In $\textsc{Encaps}(i)$, we generate the key K by independent uniform sampling from the key space $\{0,1\}^\kappa$ (Line 20), and record (ct, ψ, K) in a list $\mathcal{CK}^{\mathsf{ow}}_i$ (Line 21). When the adversary queries H_i on (ct, ψ), if (ct, ψ) is generated in $\textsc{Encaps}(i)$, then we return K where $(\mathsf{ct}, \psi, K) \in \mathcal{CK}^{\mathsf{ow}}_i$ (Lines 27 to 28).

This modification does not change the view of $\mathcal{A}$ unless $\textsc{Encaps}(i)$ generates some (ct, ψ) that $\mathcal{A}$ queried H_i on (ct, ψ). By the ciphertext entropy of KEM_0, we have

$$\left|\Pr\left[\mathbf{G}^{\mathcal{A}}_0 \Rightarrow 1\right] - \Pr\left[\mathbf{G}^{\mathcal{A}}_1 \Rightarrow 1\right]\right| \leq \frac{NQ_{\mathsf{E}}Q_{\mathsf{H}}}{2^{\gamma_0}}.$$

Game $\mathbf{G}_2$: We introduce an abort rule in H_i and modify the simulation of H_i for each $i \in [N]$:

- If $\mathcal{A}$ queries H_i on (ct, ψ), where ct is a challenge ciphertext wrt pk_i, ψ is the KEM_0 key of ct, and user i is not opened ($\mathsf{opened}[i] = 0$), then the game aborts (Lines 29 to 31). Let $\mathsf{bad}^{\mathsf{ow}}$ be the event that $\mathcal{A}$ issued such queries to H_i for some $i \in [N]$.
- When $\mathcal{A}$ queries H_i on (ct, ψ), if user i is opened ($\mathsf{opened}[i] = 1$) and ψ is the KEM_0 key of ct, then we return K where (ct, K) is recorded in $\mathcal{CK}_i$ (Lines 32 to 35).

We claim that if $\mathsf{bad}^{\mathsf{ow}}$ does not happen, then the view of $\mathcal{A}$ in $\mathbf{G}_1$ is the same as in $\mathbf{G}_2$. Suppose that $\mathsf{bad}^{\mathsf{ow}}$ does not happen in $\mathbf{G}_1$. For any unopened user i, $\mathcal{A}$ never queries the hash input (ct, ψ) of the

Game $\mathbf{G}_4$

01 $\mathsf{par} \leftarrow \mathsf{Setup}_0(1^\lambda)$
02 **for** $i \in [N]$:
03 $\quad(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{SimGen}(\mathsf{par})$
04 $\quad\mathsf{opened}[i] := 0$
05 $\quad\mathcal{D}_i := \emptyset, \mathcal{C}_i := \emptyset$
06 $\quad\mathcal{H}_i := \emptyset, \mathcal{CK}_i := \emptyset$
07 **return** $b \leftarrow \mathcal{A}^{\mathrm{O}}(\mathsf{par}, (\mathsf{pk}_i)_{i\in[N]})$

Oracle $\mathsf{H}_i(\mathsf{ct}, \psi)$ $\quad$ // $i \in [N]$

08 **if** $\exists K$ s.t. $(\mathsf{ct}, \psi, K) \in \mathcal{H}_i$: **return** K
09 $K \xleftarrow{\$} \{0,1\}^\kappa$
10 **if** $\mathsf{opened}[i]$:
11 $\quad K := \mathsf{SimHash}(\mathsf{pk}, \mathsf{sk}, \mathcal{CK}_i, \mathcal{D}_i, \mathcal{H}_i, M)$
12 **else** : $K := \mathsf{SimHash}(\mathsf{pk}, \mathsf{sk}, \mathcal{C}_i, \mathcal{D}_i, \mathcal{H}_i, M)$
13 $\mathcal{H}_i := \mathcal{H}_i \cup \{(\mathsf{ct}, \psi, K)\}$
14 **return** K

Oracle $\textsc{Encaps}(i \in [N])$

15 $\mathsf{ct} \leftarrow \mathsf{SimEncaps}(\mathsf{pk}_i, \mathsf{sk}_i)$
16 $K \xleftarrow{\$} \{0,1\}^\kappa$
17 $\mathcal{CK}_i := \mathcal{CK}_i \cup \{(\mathsf{ct}, K)\}$
18 $\mathcal{C}_i := \mathcal{C}_i \cup \{\mathsf{ct}\}$
19 **return** (ct, K)

Oracle $\textsc{Open}(i \in [N])$

20 $\mathsf{opened}_i := 1$
21 **return** sk_i

Oracle $\textsc{Decaps}(i \in [N], \mathsf{ct})$

22 **if** $\mathsf{ct} \in \mathcal{C}_i$: **return** $\perp$
23 $\mathcal{D}_i := \mathcal{D}_i \cup \{\mathsf{ct}\}$
24 $\psi' := \mathsf{Decaps}_0(\mathsf{sk}_i, \mathsf{ct})$
25 **if** $\psi' = \perp$: $K' := \perp$
26 **else** $K' := \mathsf{H}(\mathsf{ct}, \psi')$
27 **return** K'

Figure 12: Game $\mathbf{G}_4$ in the proof of Theorem 5.2. The differences to $\mathbf{G}_3$ are highlighted.

challenge keys of user i, which means that $\mathsf{H}_i(\mathsf{ct}, \psi)$ will never be defined and the code in Lines 27 to 28 will never be executed (until user i is opened). That is, the code in Lines 27 to 28 is executed only if user i is opened. By definition of $\mathcal{CK}_i^{\mathsf{ow}}$, $(\mathsf{ct}, \psi, K') \in \mathcal{CK}^{\mathsf{ow}}$ is equivalent to $(\mathsf{ct}, K') \in \mathcal{CK}_i \wedge \mathsf{Decaps}_0(\mathsf{sk}_i, \mathsf{ct}) = \psi$. So, the code in Lines 32 to 35 is a rephrasing of Lines 27 to 28. Lemma 5.3 bounds the difference between $\mathbf{G}_1$ with $\mathbf{G}_2$. For readability, we postpone the proof of Lemma 5.3 and continue the proof of Theorem 5.2.

Lemma 5.3 *With the notation and assumptions from the proof of Theorem 5.2, there is a PPT algorithm $\mathcal{B}$ with $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A})$ and*

$$\left|\Pr\left[\boldsymbol{G}_1^{\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\boldsymbol{G}_2^{\mathcal{A}} \Rightarrow 1\right]\right| \leq \Pr\left[\mathsf{bad}^{\mathsf{ow}}\right] \leq \mathsf{Adv}^{\mathsf{OW\text{-}ChCCA}}_{\mathcal{B},\mathsf{KEM}_0}(\lambda) + \delta_0,$$

where $\mathcal{B}$ is a PPT algorithm with $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A})$.

Game $\mathbf{G}_3$: We undo the abort rules defined in $\mathbf{G}_2$. The difference between $\mathbf{G}_2$ with $\mathbf{G}_3$ is that if $\mathcal{A}$ triggers such abort events, $\mathbf{G}_3$ will not abort. So, by Lemma 5.3, we have

$$\left|\Pr\left[\mathbf{G}_2^{\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\mathbf{G}_3^{\mathcal{A}} \Rightarrow 1\right]\right| \leq \mathsf{Adv}^{\mathsf{OW\text{-}ChCCA}}_{\mathcal{B},\mathsf{KEM}_0}(\lambda) + \delta_0,$$

Game $\mathbf{G}_4$: We rewrite the code of $\mathbf{G}_3$ in Figure 12 to follow the syntax of the game $\mathsf{NC}^{\mathcal{A}}_{\mathsf{sim}}(\lambda)$ (Figure 9). Specifically, in $\mathbf{G}_4$, key pairs $\{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i\in[N]}$ are generated by SimGen (which is the same as Gen), challenge ciphertexts are generated by $\mathsf{SimEncaps}$ (Figure 10), and we use $\mathsf{SimHash}$ to handle queries to random oracles $\{\mathsf{H}_i\}_{i\in[N]}$ (Figure 10).

One can check that $\mathbf{G}_4$ is equivalent to $\mathbf{G}_3$, and is the same as the game $\mathsf{NC}^{\mathcal{A}}_{\mathsf{sim}}(\lambda)$. Note that at the start of the proof we assume that there is no collision among the outputs of H_i for all $i \in [N]$ and challenge ciphertexts generated in $\textsc{Encaps}(i)$ are never been issued to $\textsc{Decaps}(i)$ before for all i. We have

$$\left|\Pr\left[\mathbf{G}_4^{\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\mathsf{NC}^{\mathcal{A}}_{\mathsf{sim}}(\lambda) \Rightarrow 1\right]\right| \leq \frac{Q_{\mathsf{H}}^2 + Q_{\mathsf{E}}^2}{2^\kappa} + \frac{NQ_{\mathsf{E}}Q_{\mathsf{D}}}{2^{\gamma_0}}.$$

By combining all the probability bounds, we obtain the statement. □

of Lemma 5.3. To bound $\Pr[\mathsf{bad}^{\mathsf{ow}}]$, we construct an adversary $\mathcal{B}$ that simulates $\mathbf{G}_2$ (Figure 11) for $\mathcal{A}$ and against $\mathsf{OW\text{-}ChCCA}$ security of KEM_0. By Definition 3.1 and Figure 3, $\mathcal{B}$ is given parameters par, N public keys, and oracle accesses to $\textsc{Enc}, \textsc{Dec}, \textsc{Corr}, \textsc{Check}$. Reduction $\mathcal{B}$ is given in Figure 13.

In Figure 13, the simulations of $\textsc{Decaps}$ and $\textsc{Open}$ are straightforward. In $\textsc{Encaps}(i)$, $\mathcal{B}$ generates challenge ciphertext by querying $\textsc{Enc}(i)$. Since $\mathcal{B}$ does not know the KEM_0 keys of challenge ciphertexts

$\mathcal{B}^{\text{ENC},\text{DEC},\text{CORR},\text{CHECK}}(\mathsf{par}, (\mathsf{pk}_i)_{i\in[N]})$

$\psi^* := \bot$
for $i \in [N]$:
  $\mathsf{opened}[i] := 0$
  $\mathcal{D}_i := \emptyset, \mathcal{C}_i := \emptyset$
  $\mathcal{H}_i := \emptyset, \mathcal{CK}_i := \emptyset$
$b \leftarrow \mathcal{A}^{\text{O}}(\mathsf{par}, (\mathsf{pk}_i)_{i\in[N]})$
return ψ^*

Oracle $\text{OPEN}(i \in [N])$

$\mathsf{opened}_i := 1$
return $\mathsf{sk}_i :=$ $\text{CORR}(i)$

Oracle $\text{DECAPS}(i \in [N], \mathsf{ct})$

if $\mathsf{ct} \in \mathcal{C}_i$: **return** $\bot$
$\mathcal{D}_i := \mathcal{D}_i \cup \{\mathsf{ct}\}$
$\psi' :=$ $\text{DEC}(i, \mathsf{ct})$
if $\psi' = \bot$: $K' := \bot$
else : $K' := \mathsf{H}(\mathsf{ct}, \psi')$
return K'

Oracle $\text{ENCAPS}(i \in [N])$

$\mathsf{ct} \leftarrow$ $\text{ENC}(i)$, $K \xleftarrow{\$} \{0,1\}^\kappa$
$\mathcal{CK}_i := \mathcal{CK}_i \cup \{(\mathsf{ct}, K)\}$
$\mathcal{C}_i := \mathcal{C}_i \cup \{\mathsf{ct}\}$
return (ct, K)

Oracle $\mathsf{H}_i(\mathsf{ct}, \psi)$ // $i \in [N]$

if $\exists K$ s.t. $(\mathsf{ct}, \psi, K) \in \mathcal{H}_i$:
  return K
$K \xleftarrow{\$} \{0,1\}^\kappa$
if $\mathsf{ct} \in \mathcal{C}_i \wedge \neg\mathsf{opened}[i]$
  $\wedge$ $\text{CHECK}(i, \mathsf{ct}, \psi)$:
  $\psi^* := \psi$ //record the solution
  abort and **return** ψ^*
if $\mathsf{opened}[i]$:
  if $\exists K'$ s.t. $(\mathsf{ct}, K') \in \mathcal{CK}_i$
    $\wedge$ $\text{CHECK}(i, \mathsf{ct}, \psi)$
    $K := K'$
$\mathcal{H}_i := \mathcal{H}_i \cup \{(\mathsf{ct}, \psi, K)\}$
return K

Figure 13: The reduction $\mathcal{B}$ in the proof of Lemma 5.3. It uses the oracles (highlighted) provided by game $\mathsf{OW\text{-}ChCCA}_{\mathsf{KEM}}(\lambda)$ to simulate $\mathbf{G}_4$.

generated via $\text{ENC}(i)$ and sk_i before $\mathcal{A}$ opens party i's secret key, to simulate $\mathbf{G}_2$, $\mathcal{B}$ uses the CHECK oracle to determine whether $\mathsf{Decaps}_0(\mathsf{sk}_i, \mathsf{ct}) = \psi$ or not.

If $\mathsf{bad}^{\mathsf{ow}}$ happens, which means that $\mathcal{A}$ queries H_i on (ct, ψ) where ψ is the KEM_0 key of a challenge ciphertext ct wrt pk_i and party i is unopened, then by the simulation of H_i in Figure 13, $\mathcal{B}$ can detect such query and get the KEM_0 key of the challenge ciphertext ct. Therefore, if $\mathsf{bad}^{\mathsf{ow}}$ happens, $\mathcal{B}$ finally outputs ψ^* such that ψ^* is the one-way solution of some challenge ciphertext ct wrt pk_i and $\mathcal{B}$ never issue $\text{CORR}(i)$. By Definition 3.1 and Figure 3, $\mathcal{B}$ wins the $\mathsf{OW\text{-}ChCCA}_{\mathsf{KEM}}(\lambda)$ game. Note that we also need to count in the correctness bound of KEM_0, since KEM_0 is imperfect, $(\mathsf{ct}, \psi) \leftarrow \mathsf{Encaps}_0(\mathsf{pk}_i)$ does not always imply $\psi = \mathsf{Decaps}_0(\mathsf{sk}_i, \mathsf{ct})$. By the argument of $\mathbf{G}_2$ in the proof of Theorem 5.2, we have

$$\left|\Pr\left[\mathbf{G}_1^{\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\mathbf{G}_2^{\mathcal{A}} \Rightarrow 1\right]\right| \leq \Pr\left[\mathsf{bad}^{\mathsf{ow}}\right] \leq \mathsf{Adv}^{\mathsf{OW\text{-}ChCCA}}_{\mathcal{B},\mathsf{KEM}_0}(\lambda) + \epsilon_0.$$

□

5.2 From OW-ChCCA to SIM-BiSO-CCA

We construct a bilateral selective-opening (i.e., SIM-BiSO-CCA) secure public-key encryption tightly from any OW-ChCCA KEM with deterministic ciphertext derivation. With KEMs in Section 3, we obtain the *first* tightly SIM-BiSO-CCA secure public-key encryption scheme. SIM-BiSO-CCA security [LYHW21] is a stronger simulation-based security notion for PKE in the multi-user setting. It models selective-opening attacks on both sender and receiver sides. Concreteley, in this notion, the adversary can learn some senders' plaintexts and the randomness used for encryption and corrupt some receivers' secret keys. SIM-BiSO-CCA security guarantees that such an adversary should not learn more than these. The formal definitions of PKE and SIM-BiSO-CCA security are given in Appendix G.

Let $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encap}, \mathsf{Decap})$ be a KEM. We construct $\mathsf{PKE} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ in Figure 14 using random oracles $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^\lambda \times \{0,1\}^\lambda$ and $\mathsf{G}\colon \{0,1\}^* \to \{0,1\}^\lambda$. We show that if KEM is OW-ChCCA secure and has deterministic ciphertext derivation, then PKE is tightly SIM-BiSO-CCA secure. The formal statement and proof are postponed to Appendix G.

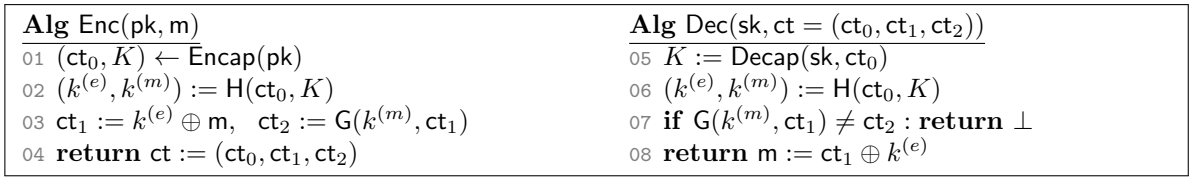

Figure 14: $\mathsf{PKE} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ from $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encap}, \mathsf{Decap})$. $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^\lambda \times \{0,1\}^\lambda$ and $\mathsf{G}\colon \{0,1\}^* \to \{0,1\}^\lambda$ are random oracles. Setup and key generation algorithms in PKE are the same as those in KEM.

References

[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 2.)

[AKPW13] Joël Alwen, Stephan Krenn, Krzysztof Pietrzak, and Daniel Wichs. Learning with rounding, revisited - new reduction, properties and applications. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 57–74. Springer, Heidelberg, August 2013. (Cited on page 6.)

[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 2, 3.)

[BHJ+15] Christoph Bader, Dennis Hofheinz, Tibor Jager, Eike Kiltz, and Yong Li. Tightly-secure authenticated key exchange. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part I*, volume 9014 of *LNCS*, pages 629–658. Springer, Heidelberg, March 2015. (Cited on page 2.)

[BLP+13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 575–584. ACM Press, June 2013. (Cited on page 6.)

[BR94] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 232–249. Springer, Heidelberg, August 1994. (Cited on page 1.)

[CCG+19] Katriel Cohn-Gordon, Cas Cremers, Kristian Gjøsteen, Håkon Jacobsen, and Tibor Jager. Highly efficient key exchange protocols with optimal tightness. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 767–797. Springer, Heidelberg, August 2019. (Cited on page 4.)

[CK01] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 453–474. Springer, Heidelberg, May 2001. (Cited on page 1.)

[CLM+18] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: An efficient post-quantum commutative group action. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part III*, volume 11274 of *LNCS*, pages 395–427. Springer, Heidelberg, December 2018. (Cited on page 5.)

[DG21] Hannah Davis and Felix Günther. Tighter proofs for the SIGMA and TLS 1.3 key exchange protocols. In Kazue Sako and Nils Ole Tippenhauer, editors, *ACNS 21, Part II*, volume 12727 of *LNCS*, pages 448–479. Springer, Heidelberg, June 2021. (Cited on page 2.)

[DJ21] Denis Diemert and Tibor Jager. On the tight security of TLS 1.3: Theoretically sound cryptographic parameters for real-world deployments. *Journal of Cryptology*, 34(3):30, July 2021. (Cited on page 2.)

[dKGV21] Bor de Kock, Kristian Gjøsteen, and Mattia Veroni. Practical isogeny-based key-exchange with optimal tightness. In Orr Dunkelman, Michael J. Jacobson, Jr., and Colin O'Flynn, editors, *Selected Areas in Cryptography*, pages 451–479, Cham, 2021. Springer International Publishing. (Cited on page 5.)

[EHK+13] Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge Villar. An algebraic framework for Diffie-Hellman assumptions. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 129–147. Springer, Heidelberg, August 2013. (Cited on page 3, 15, 27.)

[GGJJ23] Kai Gellert, Kristian Gjøsteen, Håkon Jacobsen, and Tibor Jager. On optimal tightness for key exchange with full forward secrecy via key confirmation. In *CRYPTO 2023*, LNCS. Springer, 2023. (Cited on page 17.)

[GHKW16] Romain Gay, Dennis Hofheinz, Eike Kiltz, and Hoeteck Wee. Tightly CCA-secure encryption without pairings. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 1–27. Springer, Heidelberg, May 2016. (Cited on page 27.)

[GJ18] Kristian Gjøsteen and Tibor Jager. Practical and tightly-secure digital signatures and authenticated key exchange. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 95–125. Springer, Heidelberg, August 2018. (Cited on page 2, 4.)

[GPV07] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. Cryptology ePrint Archive, Report 2007/432, 2007. https://eprint.iacr.org/2007/432. (Cited on page 6.)

[HJK+21] Shuai Han, Tibor Jager, Eike Kiltz, Shengli Liu, Jiaxin Pan, Doreen Riepel, and Sven Schäge. Authenticated key exchange and signatures with tight security in the standard model. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 670–700, Virtual Event, August 2021. Springer, Heidelberg. (Cited on page 2, 17.)

[HKSU20] Kathrin Hövelmanns, Eike Kiltz, Sven Schäge, and Dominique Unruh. Generic authenticated key exchange in the quantum random oracle model. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 389–422. Springer, Heidelberg, May 2020. (Cited on page 2, 4.)

[HLG21] Shuai Han, Shengli Liu, and Dawu Gu. Key encapsulation mechanism with tight enhanced security in the multi-user setting: Impossibility result and optimal tightness. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part II*, volume 13091 of *LNCS*, pages 483–513. Springer, Heidelberg, December 2021. (Cited on page 4, 17, 42.)

[JKRS21] Tibor Jager, Eike Kiltz, Doreen Riepel, and Sven Schäge. Tightly-secure authenticated key exchange, revisited. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 117–146. Springer, Heidelberg, October 2021. (Cited on page 2, 3, 4, 5, 16, 17, 18, 30, 41.)

[Kra05] Hugo Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 546–566. Springer, Heidelberg, August 2005. (Cited on page 3, 17.)

[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 2.)

[KYY18] Shuichi Katsumata, Shota Yamada, and Takashi Yamakawa. Tighter security proofs for GPV-IBE in the quantum random oracle model. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 253–282. Springer, Heidelberg, December 2018. (Cited on page 3, 6, 9.)

[LLGW20] Xiangyu Liu, Shengli Liu, Dawu Gu, and Jian Weng. Two-pass authenticated key exchange with explicit authentication and tight security. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 785–814. Springer, Heidelberg, December 2020. (Cited on page 2.)

[LLM07] Brian A. LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In Willy Susilo, Joseph K. Liu, and Yi Mu, editors, *ProvSec 2007*, volume 4784 of *LNCS*, pages 1–16. Springer, Heidelberg, November 2007. (Cited on page 1.)

[LSSS17] Benoît Libert, Amin Sakzad, Damien Stehlé, and Ron Steinfeld. All-but-many lossy trapdoor functions and selective opening chosen-ciphertext security from LWE. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 332–364. Springer, Heidelberg, August 2017. (Cited on page 3.)

[LYHW21] Junzuo Lai, Rupeng Yang, Zhengan Huang, and Jian Weng. Simulation-based bi-selective opening security for public key encryption. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part II*, volume 13091 of *LNCS*, pages 456–482. Springer, Heidelberg, December 2021. (Cited on page 4, 21.)

[MR04] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. In *45th FOCS*, pages 372–381. IEEE Computer Society Press, October 2004. (Cited on page 6.)

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *22nd ACM STOC*, pages 427–437. ACM Press, May 1990. (Cited on page 3.)

[OP01] 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 3, 7.)

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 333–342. ACM Press, May / June 2009. (Cited on page 6.)

[PQR21] Jiaxin Pan, Chen Qian, and Magnus Ringerud. Signed diffie-hellman key exchange with tight security. In Kenneth G. Paterson, editor, *CT-RSA 2021*, volume 12704 of *LNCS*, pages 201–226. Springer, Heidelberg, May 2021. (Cited on page 2.)

[PW22] Jiaxin Pan and Benedikt Wagner. Lattice-based signatures with tight adaptive corruptions and more. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022, Part II*, volume 13178 of *LNCS*, pages 347–378. Springer, 2022. (Cited on page 2, 17.)

[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 2, 4, 6.)

[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 2.)

Appendix

A Omitted Definitions

Definition A.1 (Ciphertext Entropy). Let $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encap}, \mathsf{Decap})$ be a key encapsulation mechanism and $h \in \mathbb{N}$. We say that KEM has h bits of ciphertext entropy if the following holds for all $\mathsf{par} \in \mathsf{Setup}(1^\lambda)$, all $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{Gen}(\mathsf{par})$, and all $\mathsf{ct}_0 \in \mathcal{C}$:

$$\Pr\left[\mathsf{ct} = \mathsf{ct}_0 \mid (\mathsf{ct}, K) \leftarrow \mathsf{Encap}(\mathsf{pk})\right] \leq 2^{-h}.$$

Definition A.2 (Public Key Entropy). Let $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encap}, \mathsf{Decap})$ be a key encapsulation mechanism and $h \in \mathbb{N}$. We say that KEM has h bits of public key entropy if the following holds for all $\mathsf{par} \in \mathsf{Setup}(1^\lambda)$, all $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{Gen}(\mathsf{par})$, and all $\mathsf{pk}_0 \in \mathcal{P}$:

$$\Pr\left[\mathsf{pk} = \mathsf{pk}_0 \mid (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{par})\right] \leq 2^{-h}.$$

B Omitted Proofs from Section 3.2

Lemma B.1 *Let $4\alpha'\alpha m < q$. Then $\mathsf{KEM}_{\mathsf{LWE}}$ is ρ-correct, for $\rho \geq 1 - \mathsf{negl}(\lambda)$.*

Proof. Let $\mathsf{par} = \mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$. Let $\mathsf{pk} = (\mathbf{U}_0, \mathbf{U}_1)$ and $\mathsf{sk} = (\mathbf{Z}_b, b)$ be output of $\mathsf{Gen}(\mathsf{par})$. Note that it is sufficient to show that $\mathbf{h}'_b = \mathbf{h}_b$, where these values are as in the definition of algorithms Encap and Decap, respectively. The rest follows easily by inspection. We have

$$\begin{aligned}\mathbf{h}'_b &= \mathsf{Round}(\hat{\mathbf{h}}_b - \mathbf{Z}_b^t\mathbf{x}) = \mathsf{Round}(\mathbf{U}_b^t\mathbf{s} + \mathbf{h}_0\lfloor q/2 \rceil - \mathbf{Z}_b^t\mathbf{x}) \\ &= \mathsf{Round}(\mathbf{Z}_b^t\mathbf{A}^t\mathbf{s} + \mathbf{h}_b\lfloor q/2 \rceil - \mathbf{Z}_b^t(\mathbf{A}^t\mathbf{s} + \mathbf{e})) = \mathsf{Round}(\mathbf{h}_b\lfloor q/2 \rceil - \mathbf{Z}_b^t\mathbf{e}).\end{aligned}$$

This is equal to $\mathbf{h}_b$, if each entry $z_i \in \mathbb{Z}_q$, $i \in [\lambda]$, of $\mathbf{Z}_b^t\mathbf{e}$ satisfies $|z_i| < q/4$. Denote column i of $\mathbf{Z}_b$ by $\mathbf{z}_{b,i} \in \mathbb{Z}_q^m$ for $i \in [\lambda]$. Then, with overwhelming probabilility, we have $\|\mathbf{z}_{b,i}\| \leq \alpha\sqrt{m}$ for all $i \in [\lambda]$ and $\|\mathbf{e}\| \leq \alpha'\sqrt{m}$. We can conclude with

$$|z_i| = |\mathbf{z}_{b,i}^t\mathbf{e}| \leq \|\mathbf{z}_{b,i}\|\|\mathbf{e}\| \leq \alpha\alpha' m \leq q/4.$$

□

of Lemma 3.4. Fix a query of the form $\textsc{Enc}(i)$. In this query, the value R is sampled uniformly at random from $\{0,1\}^\lambda$. Thus, the probability that $\mathsf{G}(R)$ is already defined before the query is at most $Q_\mathsf{G} \cdot 2^{-\lambda}$. Next, we bound the probability that $\mathcal{A}$ queries $\mathsf{G}(R)$ after this query, but before the corresponding queries of the form $\textsc{Corr}(i)$ or $\textsc{Reveal}(i, \mathsf{ct})$, where ct is the ciphertext returned by the query of interest. We call this event badR'. Note that in $\mathbf{G}_3$, as soon as event badK occurs, the game aborts. Therefore, badR' can only occur, if the adversary did neither query $\mathsf{H}(\mathbf{x}, \hat{\mathbf{h}}_0, \mathbf{h}_0)$ nor $\mathsf{H}(\mathbf{x}, \hat{\mathbf{h}}_1, \mathbf{h}_1)$ before it queries $\mathsf{G}(R)$. Note that in this case, these hash values $\hat{K}_0 := \mathsf{H}(\mathbf{x}, \hat{\mathbf{h}}_0, \mathbf{h}_0)$ and $\hat{K}_1 := \mathsf{H}(\mathbf{x}, \hat{\mathbf{h}}_1, \mathbf{h}_1)$ are uniform from $\mathcal{A}$'s point of view, which means that the values $C_0 = \hat{K}_0 \oplus R$ and $C_1 = \hat{K}_1 \oplus R$ reveal nothing about R. Thus, in each query that could trigger badR', the value R is still uniform for $\mathcal{A}$, which means that we can upper bound the probability of badR' by $Q_\mathsf{G} \cdot 2^{-\lambda}$. The claim follows. □

of Lemma 3.7. We first introduce some notation and terminology. Namely, we refer to the execution $\mathsf{Decap}((\mathbf{Z}_0, 0), \mathsf{ct})$ as the *left* execution, and to the execution $\mathsf{Decap}((\mathbf{Z}_1, 1), \mathsf{ct})$ as the *right* execution. Accordingly, when we want to refer to variables used in the left or right execution, we denote them with superscripts l and r, respectively. For example, $\mathbf{e}^l$ refers to the variable $\mathbf{e}$ that is computed in the left execution, and $\hat{K}_1^r$ refers to the variable $\hat{K}_1$ computed in the right execution. As the ciphertext $\mathsf{ct} = (C_0, C_1, \mathbf{x}, \hat{\mathbf{h}}_0, \hat{\mathbf{h}}_1)$ is the same in both executions, we omit the superscripts here.

To prove the claim, we will consider the following cases:

a) Both the left and the right execution output $\perp$.

b) The left execution outputs $R^l \neq \perp$, and the right execution outputs $\perp$.

c) The left execution outputs $R^l \neq \perp$, and the right execution outputs $R^r \neq \perp$, and we have $R^l = R^r$.

d) The left execution outputs $R^l \neq \perp$, and the right execution outputs $R^r \neq \perp$, and we have $R^l \neq R^r$.

By symmetry, this covers all the cases. Also, for the first and third case we are done. Thus, our goal is to derive a contradiction for cases b) and d).

To do that, we first show the following claims:

1. If the left execution outputs $R^l \neq \perp$, and $\hat{K}_1^l \neq \hat{K}_1^r$, then $\mathbf{h}_1'^r \neq \mathbf{h}_1^l$.
2. If the left execution outputs $R^l \neq \perp$, and $R^l \neq R^r$, then $\mathbf{h}_1'^r \neq \mathbf{h}_1^l$.
3. If the left execution outputs $R^l \neq \perp$, and $\mathbf{h}_1'^r \neq \mathbf{h}_1^l$, then we arrive at a contradiction.

Proof of Claim 1. We have $\hat{K}_1^l \neq \hat{K}_1^r$. By the definition of $\hat{K}_1$, this implies $\mathsf{H}(\mathbf{x}, \hat{\mathbf{h}}_1'^l, \mathbf{h}_1^l) \neq \mathsf{H}(\mathbf{x}, \hat{\mathbf{h}}_1, \mathbf{h}_1'^r)$. As the left execution does not return $\perp$, we know that $\hat{\mathbf{h}}_1 = \hat{\mathbf{h}}_1'^l$, and therefore it follows that $\mathbf{h}_1^l \neq \mathbf{h}_1'^r$.

Proof of Claim 2. We have $R^l \neq R^r$. Plugging in the definition of R^r, we get $R^l \neq C_1 \oplus \hat{K}_1^r$. As the left execution does not return $\perp$, we get $C_1 \oplus \hat{K}_1^l \neq C_1 \oplus \hat{K}_1^r$, which implies $\hat{K}_1^l \neq \hat{K}_1^r$. Then we can use Claim 1.

Proof of Claim 3. As the left execution does not return $\perp$, we know that $\hat{\mathbf{h}}_1 = \hat{\mathbf{h}}_1'^l$ and $\mathbf{x} = \mathbf{A}^t\mathbf{s}^l + \mathbf{e}^l$ holds. This implies that

$$\begin{aligned}\mathbf{h}_1'^r &= \mathsf{Round}(\hat{\mathbf{h}}_1 - \mathbf{Z}_1^t\mathbf{x}) = \mathsf{Round}(\hat{\mathbf{h}}_1'^l - \mathbf{Z}_1^t\mathbf{x}) = \mathsf{Round}(\mathbf{U}_1^t\mathbf{s}^l + \mathbf{h}_1^l\lfloor q/2 \rceil - \mathbf{Z}_1^t\mathbf{x}) \\ &= \mathsf{Round}(\mathbf{h}_1^l\lfloor q/2 \rceil - \mathbf{Z}_1^t\mathbf{e}^l) = \mathbf{h}_1^l,\end{aligned}$$

where the last equality follows from our assumptions about the norm of $\mathbf{Z}_1$ and $\mathbf{e}^l$. Clearly, this is a contradiction.

Contradiction for Case b). Consider the case where the left execution outputs $R^l \neq \perp$, and the right execution outputs $\perp$. We consider four sub-cases, according to the different options that let the right execution output $\perp$.

1. *Sub-Case of Case b):* $\mathbf{x} \neq \mathbf{A}^t\mathbf{s}^r + \mathbf{e}^r$. As the left execution does not output $\perp$, we know that $\mathbf{x} = \mathbf{A}^t\mathbf{s}^l + \mathbf{e}^l$. Therefore, it must hold that $R^l \neq R^r$. Using Claim 2 and then Claim 3 we get a contradiction.
2. *Sub-Case of Case b):* $\hat{K}_0^r \oplus R^r \neq C_0$. In this case, we first use the definition of R to get

   $$\hat{K}_0^r \oplus C_1 \oplus \hat{K}_1^r \neq \hat{K}_0^l \oplus R^l.$$

   As the left execution does not abort, we can derive

   $$\hat{K}_0^r \oplus C_1 \oplus \hat{K}_1^r \neq \hat{K}_0^l \oplus C_1 \oplus \hat{K}_1^l \Longrightarrow \hat{K}_0^r \oplus \hat{K}_1^r \neq \hat{K}_0^l \oplus \hat{K}_1^l.$$

   Without loss of generality, assume that $\hat{K}_1^r \neq \hat{K}_1^l$. Then, we use Claim 1 and Claim 3 to get a contradiction.
3. *Sub-Case of Case b):* $\mathbf{h}_1'^r \neq \mathbf{h}_1^r$. In this case, first assume that $\mathbf{h}_1^r = \mathbf{h}_1^l$. Then we have $\mathbf{h}_1'^r \neq \mathbf{h}_1^l$ and we can apply Claim 3 to get a contradiction. So assume that $\mathbf{h}_1^r \neq \mathbf{h}_1^l$. This implies that $R^l \neq R^r$, and we can apply Claim 2 and then Claim 3 to get a contradiction.
4. *Sub-Case of Case b):* $\hat{\mathbf{h}}_0'^r \neq \hat{\mathbf{h}}_0$ *and* $\hat{K}_0^r = R^r \oplus C_0$. As we assume no collision for H, this implies that we have $\hat{K}_0^l \neq \hat{K}_0^r$. Using the definition of R^l and the assumption made about $\hat{K}_0^r$ we get $R^l \oplus C_0 \neq R^r \oplus C_0$, which implies $R^l \neq R^r$. We can now use Claim 2 and Claim 3 to get a contradiction.

Contradiction for Case d). If both executions do not output $\perp$, but they output different $R^l \neq R^r$, then we can apply Claim 2 and then Claim 3 to obtain a contradiction. □

Alg $\mathsf{Setup}(1^\lambda)$
$(\mathbb{G}, g, p) \leftarrow \mathsf{GGen}(1^\lambda)$
$[\mathbf{A}] \xleftarrow{\$} \mathbb{G}^{(k+1)\times(k+1)}$
return $\mathsf{par} := (\mathbb{G}, g, p, [\mathbf{A}])$

Alg $\mathsf{Gen}(\mathsf{par})$
$b \xleftarrow{\$} \{0,1\},\ \mathbf{z}_b \xleftarrow{\$} \mathbb{Z}_p^{k+1}$
$[\mathbf{u}_b] := [\mathbf{A}\mathbf{z}_b],\ [\mathbf{u}_{1-b}] \xleftarrow{\$} \mathbb{G}^{k+1}$
$\mathsf{pk} := ([\mathbf{u}_0], [\mathbf{u}_1]),\ \mathsf{sk} := (\mathbf{z}_b, b)$
return $(\mathsf{pk}, \mathsf{sk})$

Alg $\mathsf{Encap}(\mathsf{pk})$
$R \xleftarrow{\$} \{0,1\}^\lambda,\ (\mathbf{s}, h_0, h_1) := \mathsf{G}(R)$
$[\mathbf{x}] := [\mathbf{A}^t\mathbf{s}] \in \mathbb{G}^{k+1}$
$\hat{h}_0 := [\mathbf{s}^t\mathbf{u}_0] \oplus h_0 \in \{0,1\}^{\log p}$
$\hat{h}_1 := [\mathbf{s}^t\mathbf{u}_1] \oplus h_1 \in \{0,1\}^{\log p}$
$\hat{K}_0 := \mathsf{H}([\mathbf{x}], \hat{h}_0, h_0),\quad C_0 := \hat{K}_0 \oplus R$
$\hat{K}_1 := \mathsf{H}([\mathbf{x}], \hat{h}_1, h_1),\quad C_1 := \hat{K}_1 \oplus R$
$\mathsf{ct} := (C_0, C_1, [\mathbf{x}], \hat{h}_0, \hat{h}_1)$
return $(\mathsf{ct}, K := R)$

Alg $\mathsf{Decap}(\mathsf{sk}, \mathsf{ct})$
let $\mathsf{ct} = (C_0, C_1, [\mathbf{x}], \hat{h}_0, \hat{h}_1)$
let $\mathsf{sk} = (\mathbf{z}_b, b)$
$h'_b := \hat{h}_b \oplus [\mathbf{z}_b^t\mathbf{x}] \in \{0,1\}^{\log p}$
$\hat{K}_b := \mathsf{H}([\mathbf{x}], \hat{h}_b, h'_b)$
$R := C_b \oplus \hat{K}_b$
$(\mathbf{s}, h_0, h_1) := \mathsf{G}(R)$
$\hat{h}'_{1-b} := [\mathbf{s}^t\mathbf{u}_{1-b}] \oplus h_1$
$\hat{K}_{1-b} := \mathsf{H}([\mathbf{x}], \hat{h}'_{1-b}, h_{1-b})$
if $\mathbf{x} \neq [\mathbf{A}^t\mathbf{s}]$: **return** $\perp$
if $\hat{K}_{1-b} \oplus R \neq C_{1-b}$: **return** $\perp$
if $h'_b \neq h_b$: **return** $\perp$
if $\hat{h}'_{1-b} \neq \hat{h}_{1-b}$: **return** $\perp$
return $K := R$

Figure 15: The key encapsulation mechanism $\mathsf{KEM}_{\mathsf{MDDH}} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encap}, \mathsf{Decap})$, where $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^\lambda$ and $\mathsf{G}\colon \{0,1\}^* \to \mathbb{Z}_p^{k+1} \times \{0,1\}^{\log p} \times \{0,1\}^{\log p}$ are random oracles.

C Postponed Construction from Matrix Decisional Diffie-Hellman

In this section, we give a construction of an OW-ChCCA secure key encapsulation mechanism based on the (matrix) decisional Diffie-Hellman assumption [EHK+13]. We first recall the assumption and necessary notation. Let GGen be a group generation algorithm, that outputs $(\mathbb{G}, g, p)$ on input 1^λ, where $\mathbb{G}$ is the description of a cyclic group of prime order $p \geq 2^\lambda$ with generator $g \in \mathbb{G}$. To represent a group element g^a for $a \in \mathbb{Z}_p$, we use the implicit notation $[a]$ following [EHK+13]. This naturally extends to matrices and vectors. Let $n, k, \ell \in \mathbb{N}$, $\mathbf{A} \in \mathbb{Z}_p^{n\times k}$, and $\mathbf{B} \in \mathbb{Z}_p^{k\times\ell}$. Then, one can compute $[\mathbf{AB}] \in \mathbb{G}^{n\times\ell}$ efficiently, given either $\mathbf{A}$ and $[\mathbf{B}]$, or $[\mathbf{A}]$ and $\mathbf{B}$. Next, we recall the k-MDDH assumption, which corresponds to the standard DDH assumption for $k = 1$.

Definition C.1 (MDDH Assumption). Let $k \in \mathbb{N}$. We say that the k-MDDH assumption holds relative to GGen, if for every PPT algorithm $\mathcal{B}$ the following advantage is negligible in λ:

$$\mathsf{Adv}^{k\text{-MDDH}}_{\mathcal{B}}(\lambda) := \left| \Pr\left[\mathcal{B}(\mathbb{G}, g, p, [\mathbf{A}], [\mathbf{b}]) = 1 \,\middle|\, \begin{array}{l}(\mathbb{G}, g, p) \leftarrow \mathsf{GGen}(1^\lambda),\\ \mathbf{A} \xleftarrow{\$} \mathbb{Z}_p^{(k+1)\times k}, \mathbf{b} \xleftarrow{\$} \mathbb{Z}_p^{k+1}\end{array}\right] \right.$$
$$\left. - \Pr\left[\mathcal{B}(\mathbb{G}, g, p, [\mathbf{A}], [\mathbf{b}]) = 1 \,\middle|\, \begin{array}{l}(\mathbb{G}, g, p) \leftarrow \mathsf{GGen}(1^\lambda), \mathbf{x} \xleftarrow{\$} \mathbb{Z}_p^k,\\ \mathbf{A} \xleftarrow{\$} \mathbb{Z}_p^{(k+1)\times k}, \mathbf{b} := \mathbf{A}\mathbf{x}\end{array}\right] \right|.$$

The assumption remains tightly equivalent if we increase the number of rows of $\mathbf{A}$, see [GHKW16].

Fix an integer $k \geq 2$. Our key encapsulation mechanism $\mathsf{KEM}_{\mathsf{MDDH}}$ is formally given in Figure 15. It makes use of random oracles $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^\lambda$ and $\mathsf{G}\colon \{0,1\}^* \to \mathbb{Z}_p^{k+1} \times \{0,1\}^{\log p} \times \{0,1\}^{\log p}$. Perfect correctness and deterministic ciphertext derivation of the scheme follows easily by inspection.

Theorem C.2 *Let* $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^\lambda$ *and* $\mathsf{G}\colon \{0,1\}^* \to \mathbb{Z}_p^{k+1} \times \{0,1\}^{\log p} \times \{0,1\}^{\log p}$ *be random oracles. If the* $(k-1)$-MDDH *assumption holds relative to* GGen*, then the scheme* $\mathsf{KEM}_{\mathsf{MDDH}}$ *is* OW-ChCCA *secure.*

Concretely, for any PPT algorithm $\mathcal{A}$ *there is a PPT algorithm* $\mathcal{B}$ *with* $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B})$ *and*

$$\mathsf{Adv}^{\mathsf{OW\text{-}ChCCA}}_{\mathcal{A},\mathsf{KEM}_{\mathsf{MDDH}}}(\lambda) \leq 12 \cdot \mathsf{Adv}^{(k-1)\text{-MDDH}}_{\mathcal{B}}(\lambda) + \mathsf{negl}(\lambda).$$

Proof. The proof is very similar to the proof of Theorem 3.3, and we only sketch the differences. The attentive reader may have noticed that the algebraic structure of LWE is only needed in the following steps.

- Change from game $\mathbf{G}_1$ to $\mathbf{G}_2$, and the similar change from $\mathbf{G}_3$ to $\mathbf{G}_4$.
- Change from game $\mathbf{G}'_j$ to $\mathbf{G}''_j$ in the proof of Lemma 3.5.
- Change from game $\mathbf{G}_4$ to $\mathbf{G}_5$ in the proof of Lemma 3.6.
- Change from game $\mathbf{G}_5$ to $\mathbf{G}_6$ in the proof of Lemma 3.6, and usage of Lemma 3.7.

Therefore, we only discuss how to make these steps work in the MDDH setting.

Game $\mathbf{G}_1$ to Game $\mathbf{G}_2$, Game $\mathbf{G}_3$ to Game $\mathbf{G}_4$: Recall that in this step in the proof of Theorem 3.3, the distribution of the matrix $\mathbf{A}$ in the public parameters is changed. Here, we do a similar change. Namely, in $\mathbf{G}_1$ and $\mathbf{G}_4$, $[\mathbf{A}]$ is generated uniformly as in the scheme, i.e. $[\mathbf{A}] \xleftarrow{\$} \mathbb{G}^{(k+1)\times(k+1)}$. In $\mathbf{G}_2$ and $\mathbf{G}_3$, matrix $\mathbf{A}$ is sampled as

$$\mathbf{B} \xleftarrow{\$} \mathbb{Z}_p^{(k+1)\times(k-1)}, \quad \mathbf{C} \xleftarrow{\$} \mathbb{Z}_p^{(k-1)\times 2}, \quad [\mathbf{A}] := [\mathbf{B}|\mathbf{BC}]\,.$$

Indistinguishability follows tightly from two applications (one for each column of $\mathbf{C}$) of the $(k-1)$-MDDH assumption.

Game $\mathbf{G}'_j$ to Game $\mathbf{G}''_j$: Recall that in this step in the proof of Lemma 3.5, the ciphertext component $\hat{\mathbf{h}}_{1-b_i}$ in the jth query to oracle Enc is changed. While it has the form $\hat{\mathbf{h}}_{1-b_i} := \mathbf{U}_{i,1-b_i}^t \mathbf{s} + \mathbf{h}_{1-b_i} \lfloor q/2 \rceil$ in $\mathbf{G}'_j$, it is switched to uniform $\hat{\mathbf{h}}_{1-b_i} \xleftarrow{\$} \mathbb{Z}_q^\lambda$ in $\mathbf{G}''_j$. Indistinguishability is argued using the generalized leftover hash lemma. Let us explain how to mimic this change. To this end, consider the jth query to oracle Enc, which is of the form $\textsc{Enc}(i)$ and outputs a ciphertext $\mathsf{ct} = (C_0, C_1, [\mathbf{x}], \hat{h}_0, \hat{h}_1)$, where

$$[\mathbf{x}] := [\mathbf{A}^t\mathbf{s}], \quad \hat{h}_{1-b_i} := [\mathbf{s}^t\mathbf{u}_{i,1-b_i}] \oplus h_{1-b_i}, \quad \hat{h}_{b_i} := [\mathbf{s}^t\mathbf{u}_{i,b_i}] \oplus h_{b_i},$$

for $(\mathbf{s}, h_0, h_1) := \mathsf{G}(R)$. We consider the component $\hat{h}_{1-b_i}$. Namely, we change its distribution in $\mathbf{G}''_j$ to uniform, i.e. $\hat{h}_{1-b_i} \xleftarrow{\$} \{0,1\}^{\log p}$. To argue indistinguishability, we use a statistical argument, exploiting the structure of matrix $[\mathbf{A}]$. Recall that in this step of the proof, the matrix has the structure $\mathbf{A} := [\mathbf{B}|\mathbf{BC}]$ for $\mathbf{B} \in \mathbb{Z}_p^{(k+1)\times(k-1)}$ and $\mathbf{C} \in \mathbb{Z}_p^{(k-1)\times 2}$. Therefore, we know that there is a 2-dimensional kernel on the left, i.e. there is a matrix $\mathbf{A}^\perp \in \mathbb{Z}_p^{(k+1)\times 2}$ such that $\mathbf{A}^t\mathbf{A}^\perp = \mathbf{0}$. Decomposing $\mathbf{s}$ into $\mathbf{s} = \mathbf{a} + \mathbf{A}^\perp\mathbf{b}$ for $\mathbf{a} \in \mathbb{Z}_p^{k+1}$ and $\mathbf{b} \in \mathbb{Z}_p^2$, we see that

$$\mathbf{x} = \mathbf{A}^t\mathbf{s} = \mathbf{A}^t\mathbf{a} + \mathbf{A}^t\mathbf{A}^\perp\mathbf{b} = \mathbf{A}^t\mathbf{a}$$

reveals no information about $\mathbf{b}$. Therefore, $\mathbf{s}$ has at least $2\log p$ bits of entropy given $\mathbf{x}$. As $\hat{h}_{b_i} \in \{0,1\}^{\log p}$, it can reveal at most $\log p$ bits about $\mathbf{s}$. Thus, $\mathbf{s}$ has at least $2\log p - \log p = \log p$ bits of entropy given $\mathbf{x}$ and $\hat{h}_{b_i}$. Therefore, we can apply the generalized leftover hash lemma (Lemma 2.5) as in the proof of Lemma 3.5, using $\mathbf{s} \mapsto \mathbf{s}^t\mathbf{u}_{i,1-b_i}$ as a universal family of hash functions.

Game $\mathbf{G}_4$ to Game $\mathbf{G}_5$: Recall that in this step in the proof of Lemma 3.6, the distribution of the matrices $\mathbf{U}_{i,1-b_i}$ is changed from uniform to $\mathbf{U}_{i,1-b_i} = \mathbf{A}\mathbf{Z}_{i,1-b_i}$. Intuitively, due to this change, the game now knows a secret key $\mathbf{Z}_{i,1-b_i}$ for both the b_i-side and the $1-b_i$-side, thereby removing information about bit b_i. We mimic this change in the MDDH setting as follows. Recall that in game $\mathbf{G}_4$, for each $i \in [N]$, the vector $[\mathbf{u}_{i,1-b_i}] \in \mathbb{G}^{k+1}$ is sampled as in the scheme, i.e. uniformly at random. Now, in game $\mathbf{G}_5$, $[\mathbf{u}_{i,1-b_i}]$ is sampled by first sampling $\mathbf{z}_{i,1-b} \xleftarrow{\$} \mathbb{Z}_p^{k+1}$, and then setting $[\mathbf{u}_{i,1-b_i}] := [\mathbf{A}\mathbf{z}_{i,1-b_i}]$. Note that in $\mathbf{G}_4$ and $\mathbf{G}_5$, matrix $\mathbf{A}$ is uniform over $\mathbb{Z}_p^{(k+1)\times(k+1)}$, and therefore, with overwhelming probability it has full rank. Assuming $\mathbf{A}$ has full rank, the vector $\mathbf{A}\mathbf{z}_{i,1-b_i}$ is uniform over $\mathbb{Z}_p^{k+1}$. This shows (statistical) indistinguishability of $\mathbf{G}_4$ and $\mathbf{G}_5$.

Game $\mathbf{G}_5$ to Game $\mathbf{G}_6$: The step from $\mathbf{G}_5$ to $\mathbf{G}_6$ in the proof of Lemma 3.6 is only necessary because Lemma 3.7 is applied in $\mathbf{G}_6$. An analogous lemma in the MDDH setting does not rely on any norm constraints and is proven similar to Lemma 3.7. Therefore, ruling out collisions of random oracle H is enough in the MDDH setting. This can be done exactly as in the proof of Lemma 3.6. □

D Security Model for AKE

We consider N parties $P_1, \ldots, P_N$ in the security game $\mathsf{wFS\text{-}St}_b$ shown in Figure 16, where each party $P_i (i \in [N])$ has a unique long-term key pair $(\mathsf{pk}'_i, \mathsf{sk}'_i)$. Each party may have multiple sessions at the same time, and each session between two parties has a unique session identification number sID and has the following variables defined relative to sID:

- $\mathtt{Init}[\mathsf{sID}] \in [N]$ denotes the initiator of the session.
- $\mathtt{Resp}[\mathsf{sID}] \in [N]$ denotes the responder of the session.
- $\mathtt{Type}[\mathsf{sID}] \in \{\text{“In”}, \text{“Re”}\}$ denotes the session is owned by the initiator or by the responder.
- $\mathtt{Used}[\mathsf{sID}]$ denotes whether the session sID was used if $\mathtt{Type}[\mathsf{sID}] = \text{“In”}$.
- $\mathtt{I}[\mathsf{sID}]$ denotes the messages that was computed by the initiator.
- $\mathtt{R}[\mathsf{sID}]$ denotes the messages that was computed by the responder.
- $\mathtt{St}[\mathsf{sID}]$ denotes the state information that is stored by the initiator.
- $\mathtt{SK}[\mathsf{sID}]$ denotes the session key of the session.

Game $\mathsf{wFS\text{-}St}_b$

$\mathsf{cnt} := 0, \mathcal{S} := \emptyset$
$\mathsf{par}' \leftarrow \mathsf{Setup}(1^\lambda)$
for $i \in [N] : (\mathsf{pk}'_i, \mathsf{sk}'_i) \leftarrow \mathsf{KG}'(\mathsf{par}')$
$\mathrm{O}_1 := (\textsc{Session}_\mathrm{I}, \textsc{Der}_\mathrm{I}, \textsc{Session}_\mathrm{R})$
$\mathrm{O}_2 := (\textsc{Cor}, \textsc{RevK}, \textsc{RevSt}, \textsc{Test})$
$b' \leftarrow \mathcal{A}^{\mathrm{O}_1, \mathrm{O}_2}(\mathsf{par}', (\mathsf{pk}'_i)_{i \in [N]})$
for $\mathsf{sID}^* \in \mathcal{S}$
  if $\mathsf{Fresh}(\mathsf{sID}^*) = 0 \vee \mathsf{Valid}(\mathsf{sID}^*) = 0$:
    return 0
return b'

Oracle $\textsc{Test}(\mathsf{sID})$

if $\mathsf{sID} \in \mathcal{S} \vee \mathtt{SK}[\mathsf{sID}] = \perp$: **return** $\perp$
$\mathcal{S} := \mathcal{S} \cup \{\mathsf{sID}\}$
$\mathsf{SK}_0 := \mathtt{SK}[\mathsf{sID}]$, $\mathsf{SK}_1 \xleftarrow{\$} \mathcal{SK}$
return SK_b

Oracle $\textsc{RevK}(\mathsf{sID})$

$\mathtt{revSK}[\mathsf{sID}] := 1$
return $\mathtt{SK}[\mathsf{sID}]$

Oracle $\textsc{RevSt}(\mathsf{sID})$

if $\mathtt{Type}[\mathsf{sID}] \neq \text{“In”}$: **return** $\perp$
$\mathtt{revST}[\mathsf{sID}] := 1$
return $\mathtt{St}[\mathsf{sID}]$

Oracle $\textsc{Session}_\mathrm{I}((i,j) \in [N]^2)$

$\mathsf{cnt} := \mathsf{cnt} + 1$, $\mathsf{sID} := \mathsf{cnt}$
$(\mathtt{Init}[\mathsf{sID}], \mathtt{Resp}[\mathsf{sID}]) := (i, j)$
$\mathtt{Type}[\mathsf{sID}] := \text{“In”}$
$(\mathsf{M}_i, \mathsf{st}) := \mathsf{Init}(\mathsf{sk}_i, \mathsf{pk}_j, \mathsf{par}')$
$(\mathtt{I}[\mathsf{sID}], \mathtt{St}[\mathsf{sID}]) := (\mathsf{M}_i, \mathsf{st})$
return $(\mathsf{sID}, \mathsf{M}_i)$

Oracle $\textsc{Der}_\mathrm{I}(\mathsf{sID}, \mathsf{M})$

if $\mathtt{Used}[\mathsf{sID}] = 1 \vee \mathtt{St}[\mathsf{sID}] = \perp$
  $\vee \mathtt{SK}[\mathsf{sID}] \neq \perp$: **return** $\perp$
$\mathtt{Used}[\mathsf{sID}] := 1$, $\mathsf{st} := \mathtt{St}[\mathsf{sID}]$
$(i, j) := (\mathtt{Init}[\mathsf{sID}], \mathtt{Resp}[\mathsf{sID}])$
$\mathsf{SK} := \mathsf{Der}_\mathsf{I}(\mathsf{sk}'_i, \mathsf{pk}'_j, \mathsf{M}, \mathsf{st})$
$(\mathtt{R}[\mathsf{sID}], \mathtt{SK}[\mathsf{sID}]) := (\mathsf{M}, \mathsf{SK})$
return 1

Oracle $\textsc{Session}_\mathrm{R}((i,j) \in [N]^2, \mathsf{M})$

$\mathsf{cnt} := \mathsf{cnt} + 1$, $\mathsf{sID} := \mathsf{cnt}$
$(\mathtt{Init}[\mathsf{sID}], \mathtt{Resp}[\mathsf{sID}]) := (i, j)$
$\mathtt{Type}[\mathsf{sID}] := \text{“Re”}$
$(\mathsf{M}_j, \mathsf{SK}) := \mathsf{Der}_\mathsf{R}(\mathsf{sk}'_j, \mathsf{pk}'_i, \mathsf{M})$
$(\mathtt{I}[\mathsf{sID}], \mathtt{R}[\mathsf{sID}]) := (\mathsf{M}, \mathsf{M}_j)$
$\mathtt{SK}[\mathsf{sID}] := \mathsf{SK}$
return $(\mathsf{sID}, \mathsf{M}_j)$

Oracle $\textsc{Cor}(i)$

$\mathtt{Cor}[i] := 1$
return sk'_i

Figure 16: The game $\mathsf{wFS\text{-}St}_b$ for an AKE protocol $\mathsf{AKE} = (\mathsf{Setup}', \mathsf{KG}', \mathsf{Init}, \mathsf{Der}_\mathsf{R}, \mathsf{Der}_\mathsf{I})$. This paper only considers two-message AKE protocols and in such protocols, the responder does not need to store session state to compute the session key. Therefore, in this model, $\textsc{RevSt}(\mathsf{sID})$ returns $\perp$ if sID is not an initiator session.

To capture the adversary's ability to control the channel, in the security game as described in Figure 16, adversary $\mathcal{A}$ is given access to oracles $\textsc{Session}_\mathrm{I}, \textsc{Session}_\mathrm{R}$, and $\textsc{Der}_\mathrm{I}$. $\textsc{Session}_\mathrm{I}$ creates a session owned by the initiator, and $\textsc{Session}_\mathrm{R}$ creates a session owned by the responder, which are different sessions. More precisely, $\mathcal{A}$ queries $\textsc{Session}_\mathrm{I}(i,j)$ to activate a session between P_i (as initiator) with P_j (as responder) and gets $(\mathsf{sID}, \mathsf{M}_i)$, where sID is the identification number of this session and M_i is P_i's initiator protocol message (generated from Init). The query $\textsc{Session}_\mathrm{R}(i, j, \mathsf{M})$ captures the process that $\mathcal{A}$ sends M to P_j, activates a session between P_i (as initiator) with P_j (as responder), and gets the identification number

sID and the responded protocol message M_j of this session. To complete a session sID owned by an initiator, $\mathcal{A}$ queries $\textsc{Der}_{\mathrm{I}}(\mathsf{sID}, \mathsf{M})$, which capturs the process that $\mathcal{A}$ sends the responded message M to the initiator of the session. Moreover, $\textsc{Der}_{\mathrm{I}}$ and $\textsc{Session}_{\mathrm{R}}$ may output $\perp$ to indicate that the session does not generate a session key.

In $\mathsf{wFS\text{-}St}_b$, $\mathcal{A}$ can forward messages between sessions honestly, or modify messages to launch some attack. Let sID and sID' be two sessions. We define two relationships between two sessions.

Definition D.1 ((Partially) Matching Session). We say sessions sID and sID' match if the same parties are involved (*i.e.*, $(\mathtt{Init}[\mathsf{sID}], \mathtt{Resp}[\mathsf{sID}]) = (\mathtt{Init}[\mathsf{sID}'], \mathtt{Resp}[\mathsf{sID}'])$), the messages sent and received are the same $((\mathtt{I}[\mathsf{sID}], \mathtt{R}[\mathsf{sID}]) = (\mathtt{I}[\mathsf{sID}'], \mathtt{R}[\mathsf{sID}']))$, and they are of different types $\mathtt{Type}[\mathsf{sID}] \neq \mathtt{Type}[\mathsf{sID}']$.

We say sID is partially matching to sID' if $\mathtt{Type}[\mathsf{sID}] =$ "In", $\mathtt{Type}[\mathsf{sID}'] =$ "Re", and the initial messages are the same $(\mathtt{I}[\mathsf{sID}] = \mathtt{I}[\mathsf{sID}'])$.

Sessions sID and sID' match means that $\mathcal{A}$ honestly delivers the protocol message between sID with sID' without any modification. If $\mathtt{Type}[\mathsf{sID}] =$ "In", sID partially matches sID' means that $\mathcal{A}$ honestly sends the protocol message generated from sID to sID'.

Furthermore, $\mathcal{A}$ has access to oracles $\textsc{Cor}, \textsc{RevK}$, and $\textsc{RevSt}$ to reveal secret information. By querying $\textsc{Cor}(i)$, $\mathcal{A}$ can get the long-term secret key of party P_i. $\mathcal{A}$ can also obtain the state information or the session key of session sID by querying $\textsc{RevSt}(\mathsf{sID})$ or $\textsc{RevK}(\mathsf{sID})$, respectively. We use the following variables to keep track of which queries the adversary made and which secret information is revealed. If a variable is 1, then it means the corresponding secret information is revealed.

- $\mathtt{Cor}[i]$ denotes whether the long-term secret key of party P_i was revealed.
- $\mathtt{revSK}[\mathsf{sID}]$ denotes whether the session key of session sID was revealed.
- $\mathtt{revST}[\mathsf{sID}]$ denotes whether the state information of session sID was revealed.

Finally, $\mathcal{A}$ is given access to oracle $\textsc{Test}$ which will return either the session key of the given session or a uniformly random key independent of the given session. $\mathcal{A}$ is allowed to queries $\textsc{Test}$ multiple times. A session sID that has been queried to $\textsc{Test}$ is called as a test session. All test sessions are stored in a set $\mathcal{S}$. $\mathcal{A}$'s task in this model is to distinguish the keys output by $\textsc{Test}$ are either the actual session keys of test sessions or independent random keys. To avoid trivial attack, we define two properties, freshness and validity, that all test sessions have to satisfy:

Definition D.2 (Freshness and Validity). Let sID^* be a session. sID^* is fresh if the session key of sID^* was not revealed, and if sID^* has a matching session, we also require that the matching session was not a test session and its session key was not revealed. The process to determine freshness is given in Figure 17.

Furthermore, a test session is valid if it is fresh and $\mathcal{A}$ performed any attack which is defined in this AKE model. We capture all valid attacks in Table 1, and the process to determine validity is given in Figure 17.

In Table 1, all attacks are defined using the boolean variables that indicates which queries the adversary made. This table is obtained by considering all possible attacks and excluding all trivial or redundant attacks. For a full attack table, please refer to [JKRS21]. Informally, the attacks defined in Table 1 capture weak forward secrecy (wFS), state reveal attack, and key compromise impersonation (KCI) combined with state reveal attack. Moreover, if the AKE protocol does not use appropriate randomness, i.e., $\mathcal{A}$ is able to create more than one (partially) matching session to a test session, then it is insecure in this model.

In this model, we require that every test session is fresh and valid. The adversary wins if it distinguishes the session keys from uniformly random keys which it obtains through queries to the Test oracle.

Definition D.3 (Key Indistinguishability of AKE). Let $\mathsf{AKE} := (\mathsf{Setup}', \mathsf{KG}', \mathsf{Init}, \mathsf{Der}_{\mathsf{R}}, \mathsf{Der}_{\mathsf{I}})$ be an AKE protocol and consider the games $\mathsf{wFS\text{-}St}_b$ for $b \in \{0, 1\}$ defined in Figure 16. We say AKE is $\mathsf{wFS\text{-}St}$ secure, if for all PPT adversaries $\mathcal{A}$, the following advantage is negligible:

$$\mathsf{Adv}^{\mathsf{wFS\text{-}St}}_{\mathcal{A},\mathsf{AKE}}(\lambda) := \left| \Pr\left[\mathsf{wFS\text{-}St}^{\mathcal{A}}_0(\lambda) \Rightarrow 1\right] - \Pr\left[\mathsf{wFS\text{-}St}^{\mathcal{A}}_1(\lambda) \Rightarrow 1\right] \right|$$

Alg $\mathsf{Fresh}(\mathsf{sID}^*)$

$(i^*, j^*) := (\mathtt{Init}[\mathsf{sID}^*], \mathtt{Resp}[\mathsf{sID}^*])$

$\mathfrak{M}(\mathsf{sID}^*) := \{\mathsf{sID} |\ (\mathtt{Init}[\mathsf{sID}], \mathtt{Resp}[\mathsf{sID}]) = (i^*, j^*)$
$\wedge\ (\mathtt{I}[\mathsf{sID}], \mathtt{R}[\mathsf{sID}]) = (\mathtt{I}[\mathsf{sID}^*], \mathtt{R}[\mathsf{sID}^*])$
$\wedge\ \mathtt{Type}[\mathsf{sID}] \neq \mathtt{Type}[\mathsf{sID}^*]\}$ //Matching session(s) of sID^*

if $\mathtt{revSK}[\mathsf{sID}^*] \vee (\exists \mathsf{sID} \in \mathfrak{M}[\mathsf{sID}^*] : \mathtt{revSK}[\mathsf{sID}] = 1)$: **return** 0
//$\mathcal{A}$ trivially learned the test session's key

if $\exists \mathsf{sID} \in \mathfrak{M}(\mathsf{sID}^*)$ s.t. $\mathsf{sID} \in \mathcal{S}$: **return** 0 //$\mathcal{A}$ also tested a matching session

return 1

Alg $\mathsf{Valid}(\mathsf{sID}^*)$

$(i^*, j^*) := (\mathtt{Init}[\mathsf{sID}^*], \mathtt{Resp}[\mathsf{sID}^*])$

$\mathfrak{M}(\mathsf{sID}^*) := \{\mathsf{sID} |\ (\mathtt{Init}[\mathsf{sID}], \mathtt{Resp}[\mathsf{sID}]) = (i^*, j^*)$
$\wedge\ (\mathtt{I}[\mathsf{sID}], \mathtt{R}[\mathsf{sID}]) = (\mathtt{I}[\mathsf{sID}^*], \mathtt{R}[\mathsf{sID}^*])$
$\wedge\ \mathtt{Type}[\mathsf{sID}] \neq \mathtt{Type}[\mathsf{sID}^*]\}$ //Matching session(s) of sID^*

$\mathfrak{P}(\mathsf{sID}^*) := \{\mathsf{sID} |\ \mathtt{I}[\mathsf{sID}] = \mathtt{I}[\mathsf{sID}^*]$
$\wedge\ \mathtt{Type}[\mathsf{sID}] \neq \mathtt{Type}[\mathsf{sID}^*]$
$\wedge\ \mathtt{Type}[\mathsf{sID}] = \text{“In”}\}$ //Partially matching session(s) of sID^*

if the attack type of $\mathsf{sID}^* \in$ Table 1: **return** 1

else return 0

Figure 17: Algorithms to check the validity and freshness of tested sessions.

E Security Proofs for Our AKE Protocol in Figure 8

In this section, we show the tight security of our (direct) AKE protocol (from Figure 8). More precisely, we show Theorem 4.3.

Theorem 4.3. Let $\mathcal{A}$ be an adversary against AKE in the $\mathsf{wFS\text{-}St}_b$ game, where N is the number of parties. Let S be the number of sessions in the game and T be the number of TEST queries issued by $\mathcal{A}$. Without loss of generality, we assume that every integer $i \in [N]$ is associated with the public key $\mathsf{pk}'_i (= \mathsf{pk}_i)$ of party i. The game sequences of the proof are given in Figure 18 and Figure 19. Our goal is to bound

$$\mathsf{Adv}^{\mathsf{wFS\text{-}St}}_{\mathcal{A},\mathsf{AKE}}(\lambda) = \left|\Pr\left[\mathsf{wFS\text{-}St}^{\mathcal{A}}_{0,\mathsf{AKE}} \Rightarrow 1\right] - \Pr\left[\mathsf{wFS\text{-}St}^{\mathcal{A}}_{1,\mathsf{AKE}} \Rightarrow 1\right]\right|$$

Game $\mathbf{G}_{0,b}$: This game is the same as $\mathsf{wFS\text{-}St}_{\mathsf{AKE},b}$, except that we exclude collisions of long-term key pairs $(\mathsf{pk}_i, \mathsf{sk}_i)$, ciphertexts ct_i from KEM_1, ephemeral key pairs $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}})$ and ciphertexts $\widetilde{\mathsf{ct}}$ from KEM_0, and the output of H and G. If such a collision happens at any time, then we abort the game. For readability, we do not explicitly define such events in the game.

We supposed that KEM_1 (resp., KEM_0) has γ_1 (resp., γ_0) bits ciphertext entropy and μ_1 (resp., μ_0) bits public key entropy, so by union bound and birthday bound, excluding such collision events will add $N^2 \cdot \frac{1}{2^{\mu_1}} + S^2 \cdot (\frac{1}{2^{\gamma_1}} + \frac{1}{2^{\gamma_0}} + \frac{1}{2^{\mu_0}}) + (Q^2_{\mathsf{H}} + S^2)/|\mathcal{SK}| + (Q^2_{\mathsf{G}} + N^2 + S^2)/2^d$ to the bound of Theorem 4.3. So, we have

$$\begin{aligned}&\left|\Pr\left[\mathsf{wFS\text{-}St}^{\mathcal{A}}_{b,\mathsf{AKE}} \Rightarrow 1\right] - \Pr\left[\mathbf{G}^{\mathcal{A}}_{0,b} \Rightarrow 1\right]\right| \\ &\leq \frac{N^2}{2^{\mu_1}} + S^2 \cdot (\frac{1}{2^{\gamma_1}} + \frac{1}{2^{\gamma_0}} + \frac{1}{2^{\mu_0}}) + \frac{Q^2_{\mathsf{H}} + S^2}{|\mathcal{SK}|} + \frac{Q^2_{\mathsf{G}} + N^2 + S^2}{2^d}\end{aligned}$$

Note that excluding such collisions means that for any $\mathsf{sID}^* \in \mathcal{S}$, sID^* does not fall into the attack case (0) in Table 1, since different executions of SESSION_I will output different protocol messages (i.e., different ctxt), so it is impossible for a session to have more than one partial matching sessions.

Game $\mathbf{G}_{1,b}$: We modify the simulation of generating session state and session key. In this game, we generate session keys and session states by uniformly sampling (instead of using H and G), and then patch H and G if $\mathcal{A}$ queries the corresponding hash inputs of session keys and session states. Concretely,

Game $\mathbf{G}_{0,b}$-$\mathbf{G}_{4,b}$

$\mathsf{cnt} := 0, \mathcal{S} := \emptyset$
$\mathsf{bad}_{\mathsf{st}} := 0$ // $\mathbf{G}_{2,b}$-$\mathbf{G}_{4,b}$
$\mathsf{bad}_1 := 0$ // $\mathbf{G}_{3,b}$-$\mathbf{G}_{4,b}$
$\mathsf{bad}_0 := 0$ // $\mathbf{G}_{4,b}$
$\mathsf{par}' := (\widetilde{\mathsf{par}}, \mathsf{par}) \leftarrow \mathsf{Setup}'(1^\lambda)$
for $i \in [N]$:
  $(\mathsf{pk}_i, (\mathsf{sk}_i, \mathsf{k}_i)) \leftarrow \mathsf{KG}'(\mathsf{par}')$
  $\mathcal{C}_i := \emptyset$ // $\mathbf{G}_{3,b}$-$\mathbf{G}_{4,b}$
$b' \leftarrow \mathcal{A}^{\mathsf{H},\mathsf{G},\mathrm{O}_1,\mathrm{O}_2}(\mathsf{par}', (\mathsf{pk}_i)_{i\in[N]})$
for $\mathsf{sID}^* \in \mathcal{S}$
  if $\mathsf{Fresh}(\mathsf{sID}^*) = 0$
    $\vee\ \mathsf{Valid}(\mathsf{sID}^*) = 0$:
    return 0
return b'

Oracle $\textsc{Session}_{\mathrm{I}}((i,j) \in [N]^2)$

$\mathsf{cnt} := \mathsf{cnt} + 1, \mathsf{sID} := \mathsf{cnt}$
$(\mathtt{Init}[\mathsf{sID}], \mathtt{Resp}[\mathsf{sID}]) := (i,j)$
$\mathtt{Type}[\mathsf{sID}] :=$ "In"
$(\mathsf{ct}_j, \mathsf{K}_j) \leftarrow \mathsf{Encaps}_1(\mathsf{pk}_j)$
if $\neg\mathtt{Cor}[j]$: // $\mathbf{G}_{3,b}$-$\mathbf{G}_{4,b}$
  $\mathcal{C}_j := \mathcal{C}_j \cup \{\mathsf{ct}_j\}$ // $\mathbf{G}_{3,b}$-$\mathbf{G}_{4,b}$
$(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}) \leftarrow \mathsf{KG}_0(\widetilde{\mathsf{par}})$
$IV \xleftarrow{\$} \{0,1\}^\kappa$
$\widetilde{\mathcal{K}} := \widetilde{\mathcal{K}} \cup \{(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}})\}$ // $\mathbf{G}_{1,b}$-$\mathbf{G}_{4,b}$
$\mathtt{Cor}'[\widetilde{\mathsf{pk}}] := 0$ // $\mathbf{G}_{4,b}$
$\mathsf{st}' := (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}, \mathsf{ct}_j, \mathsf{K}_j)$
$\mathsf{st} := (IV, \mathsf{G}(IV, \mathsf{k}_i) \oplus \mathsf{st}')$ // $\mathbf{G}_{0,b}$
$\varphi \xleftarrow{\$} \{0,1\}^d$ // $\mathbf{G}_{1,b}$-$\mathbf{G}_{4,b}$
$\mathsf{st} := (IV, \varphi)$ // $\mathbf{G}_{1,b}$-$\mathbf{G}_{4,b}$
$\mathtt{St}'_i[IV] := (\varphi, \mathsf{st}')$ // $\mathbf{G}_{1,b}$-$\mathbf{G}_{4,b}$
$\mathsf{M}_i := (\widetilde{\mathsf{pk}}, \mathsf{ct}_j)$
$(\mathtt{I}[\mathsf{sID}], \mathtt{St}[\mathsf{sID}]) := (\mathsf{M}_i, \mathsf{st})$
return $(\mathsf{sID}, \mathsf{M}_i)$

Oracle $\textsc{Session}_{\mathrm{R}}((i,j) \in [N]^2, \mathsf{M})$

$\mathsf{cnt} := \mathsf{cnt} + 1, \mathsf{sID} := \mathsf{cnt}$
$(\mathtt{Init}[\mathsf{sID}], \mathtt{Resp}[\mathsf{sID}]) := (i,j)$
$\mathtt{Type}[\mathsf{sID}] :=$ "Re"
let $(\widetilde{\mathsf{pk}}, \mathsf{ct}_j) := \mathsf{M}$
$\mathsf{K}_j := \mathsf{Decaps}_1(\mathsf{sk}'_j, \mathsf{ct}_j)$
if $\mathsf{K}_j = \perp$: **return** $\perp$
$(\widetilde{\mathsf{ct}}, \widetilde{\mathsf{K}}) \leftarrow \mathsf{Encaps}_0(\widetilde{\mathsf{pk}})$
$(\mathsf{ct}_i, \mathsf{K}_i) \leftarrow \mathsf{Encaps}_1(\mathsf{pk}_i)$
if $(\widetilde{\mathsf{pk}}, \cdot) \notin \widetilde{\mathcal{K}}$: // $\mathbf{G}_{1,b}$-$\mathbf{G}_{4,b}$
  $\widetilde{\mathcal{L}} := \widetilde{\mathcal{L}} \cup \{(\widetilde{\mathsf{ct}}, \widetilde{\mathsf{K}})\}$ // $\mathbf{G}_{1,b}$-$\mathbf{G}_{4,b}$
if $\neg\mathtt{Cor}[i]$: $\mathcal{C}_i := \mathcal{C}_i \cup \{\mathsf{ct}_i\}$ // $\mathbf{G}_{3,b}$-$\mathbf{G}_{4,b}$
if $\exists\widetilde{\mathsf{sk}}$ s.t. $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}) \in \widetilde{\mathcal{K}}$ // $\mathbf{G}_{4,b}$
  $\widetilde{\mathcal{C}} := \widetilde{\mathcal{C}} \cup \{(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{ct}})\}$ // $\mathbf{G}_{4,b}$
$\mathsf{ctxt} := (\mathsf{pk}_i, \mathsf{pk}_j, \widetilde{\mathsf{pk}}, \mathsf{ct}_i, \mathsf{ct}_j, \widetilde{\mathsf{ct}})$
$\mathsf{SK} := \mathsf{H}(\mathsf{ctxt}, \mathsf{K}_i, \mathsf{K}_j, \widetilde{\mathsf{K}})$ // $\mathbf{G}_{0,b}$
$\mathtt{SK}'[\mathsf{ctxt}] := \mathsf{SK} \xleftarrow{\$} \mathcal{SK}$ // $\mathbf{G}_{1,b}$-$\mathbf{G}_{4,b}$
$\mathtt{SK}[\mathsf{sID}] := \mathsf{SK}$, $\mathsf{M}_j := (\widetilde{\mathsf{ct}}, \mathsf{ct}_i)$
$(\mathtt{I}[\mathsf{sID}], \mathtt{R}[\mathsf{sID}]) := (\mathsf{M}, \mathsf{M}_j)$
return $(\mathsf{sID}, \mathsf{M}_j)$

Oracle $\textsc{Der}_{\mathrm{I}}(\mathsf{sID}, \mathsf{M})$

if $\mathtt{Used}[\mathsf{sID}] = 1 \vee \mathtt{St}[\mathsf{sID}] = \perp$
  $\vee\mathtt{SK}[\mathsf{sID}] \neq \perp$: **return** $\perp$
$\mathtt{Used}[\mathsf{sID}] := 1$
$(i,j) := (\mathtt{Init}[\mathsf{sID}], \mathtt{Resp}[\mathsf{sID}])$
let $(IV, \varphi) := \mathtt{St}[\mathsf{sID}]$
$\mathsf{st}' := \varphi \oplus \mathsf{G}(IV, \mathsf{k}_i)$ // $\mathbf{G}_{0,b}$
$(\varphi, \mathsf{st}') := \mathtt{St}'_i[IV]$ // $\mathbf{G}_{1,b}$-$\mathbf{G}_{4,b}$
let $(\widetilde{\mathsf{ct}}, \mathsf{ct}) := \mathsf{M}$, $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}, \mathsf{ct}_j, \mathsf{K}_j) := \mathsf{st}'$
$\mathsf{K}_i := \mathsf{Decaps}_1(\mathsf{sk}_i, \mathsf{ct})$
$\widetilde{\mathsf{K}} := \mathsf{Decaps}_0(\widetilde{\mathsf{sk}}, \widetilde{\mathsf{ct}})$
if $\mathsf{K}_i = \perp \vee \widetilde{\mathsf{K}} = \perp$: **return** $\perp$
$\mathsf{ctxt} := (\mathsf{pk}_i, \mathsf{pk}_j, \widetilde{\mathsf{pk}}, \mathsf{ct}_i, \mathsf{ct}_j, \widetilde{\mathsf{ct}})$
$\mathsf{SK} := \mathsf{H}(\mathsf{ctxt}, \mathsf{K}_i, \mathsf{K}_j, \widetilde{\mathsf{K}})$ // $\mathbf{G}_{0,b}$
if $\mathtt{SK}'[\mathsf{ctxt}] \neq \perp$: // $\mathbf{G}_{1,b}$-$\mathbf{G}_{4,b}$
  $\mathsf{SK} := \mathtt{SK}'[\mathsf{ctxt}]$ // $\mathbf{G}_{1,b}$-$\mathbf{G}_{4,b}$
else $\mathtt{SK}'[\mathsf{ctxt}] := \mathsf{SK} \xleftarrow{\$} \mathcal{SK}$ // $\mathbf{G}_{1,b}$-$\mathbf{G}_{4,b}$
$(\mathtt{R}[\mathsf{sID}], \mathtt{SK}[\mathsf{sID}]) := (\mathsf{M}, \mathsf{SK})$
return 1

Figure 18: The games $\mathbf{G}_{0,b}$-$\mathbf{G}_{4,b}(b \in \{0,1\})$ in the proof of Theorem 4.3. Besides H and G, $\mathcal{A}$ also has access to oracles $\mathrm{O}_1 := (\textsc{Session}_{\mathrm{I}}, \textsc{Der}_{\mathrm{I}}, \textsc{Session}_{\mathrm{R}})$ and $\mathrm{O}_2 := (\textsc{Cor}, \textsc{RevK}, \textsc{RevSt}, \textsc{Test})$. Oracles G, H, and $\textsc{RevSt}$ are shown in Figure 19. Oracles $\textsc{Cor}$, $\textsc{RevK}$, and $\textsc{Test}$ are the same as in Figure 16. Highlighted lines are only executed in the corresponding games.

$\mathcal{A}$ gets (Initiator, Responder), attack type	$\mathtt{Cor}[i^*]$	$\mathtt{Cor}[j^*]$	$\mathtt{Type}[\mathsf{sID}^*]$	$\mathtt{revST}[\mathsf{sID}^*]$	$\exists \mathsf{sID} \in \mathfrak{M}(\mathsf{sID}^*)$: $\mathtt{revST}[\mathsf{sID}]$	$\lvert\mathfrak{M}(\mathsf{sID}^*)\rvert$	$\exists \mathsf{sID} \in \mathfrak{P}(\mathsf{sID}^*)$: $\mathtt{revST}[\mathsf{sID}]$	$\lvert\mathfrak{P}(\mathsf{sID}^*)\rvert$
(0) **multiple (partially) matching sessions**	-	-	-	-	-	-	-	> 1
(1) **(long-term, long-term)**, wFS	-	-	-	**F**	**F**	1	-	-
(2) **(state, long-term)**, state attack	**F**	-	-	-	-	1	-	-
(3) **(long-term, long-term)**, wFS	-	-	"Re"	**F**	n/a	0	**F**	1
(4) **(state, long-term)**, state attack	**F**	-	"Re"	**F**	n/a	0	-	1
(5) **(state, state)**, state attack	**F**	**F**	"In"	-	n/a	0	n/a	0
(6) **(long-term, state)**, KCI + state attack	-	**F**	"In"	**F**	n/a	0	n/a	0
(7) **(state, long-term)**, KCI + state attack	**F**	-	"Re"	**F**	n/a	0	n/a	0

Table 1: List of valid attack types against two-message AKE protocols in the $\mathsf{wFS\text{-}St}_b$ game (in Figure 16). An attack is regarded as an AND conjunction of variables with specified values as shown in the each line, where **F** means "false", "-" means that the variable can take arbitrary value (can be viewed as "true"), and "n/a" means that there is no (partially) matching session exists. The sets $\mathfrak{M}(\mathsf{sID}^*)$ and $\mathfrak{P}(\mathsf{sID}^*)$ are defined in Valid procedure in Figure 17. Here "0" and "1" are numbers instead of boolean values.

- In $\textsc{Session}_{\mathrm{I}}(i, j)$, to generate $\mathtt{St}[\mathsf{sID}]$, we sample φ at uniformly random (instead of computing $\mathsf{G}(IV, \mathsf{k}_i) \oplus \mathsf{st}'$) and set (IV, φ) as the state of sID (Lines 27 to 29). We also use an internal list $\mathtt{St}'_i$ to record $(IV, \varphi, \mathsf{st}')$ (Line 29), which will be used to patch random oracle G later. Moreover, after generating $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}})$, we record it in list $\widetilde{\mathcal{K}}$ (Line 23) which are indexed by one-time vectors IV (since a session sID has at most one IV corresponding). In $\textsc{Der}_{\mathrm{I}}$, we use lists $\{\mathtt{St}'_i\}_{i\in[N]}$ to recover session state values (Line 58).
- If $\mathcal{A}$ queries $\mathsf{G}(\mathsf{k}, IV)$, k is the second tuple of party i's secret key (i.e., $\mathsf{k} = \mathsf{k}_i$), and IV corresponds a session of party i, then we use the information of $\mathtt{St}'_i$ to patch $\mathsf{G}(\mathsf{k}_i, IV)$ such that $\mathsf{G}(\mathsf{k}_i, IV) \oplus \mathsf{st}' = \varphi$ (Lines 03 to 05) to make the simulation consistent.
- In $\textsc{Session}_{\mathrm{R}}$ and $\textsc{Der}_{\mathrm{I}}$, we sample the session key uniformly at random without using H, and use an internal list $\mathtt{SK}'$ to record the transcript of the session and its corresponding session key (Line 48 and Line 65). $\mathtt{SK}'$ will be used to patch H to make the simulation consistent. We also use an internal list $\widetilde{\mathcal{L}}$ to record $(\widetilde{\mathsf{ct}}, \widetilde{\mathsf{K}})$ if $\widetilde{\mathsf{ct}}$ is not generated in $\textsc{Session}_{\mathrm{I}}$ (that is, $(\widetilde{\mathsf{pk}}, \cdot) \notin \widetilde{\mathcal{K}}$, Lines 41 to 42), since we do not have the corresponding secret key of $\widetilde{\mathsf{pk}}$ if $\widetilde{\mathsf{ct}}$ is not generated in $\textsc{Session}_{\mathrm{I}}$. In $\textsc{Der}_{\mathrm{I}}$, if session sID has a matching session, then these two sessions is set to be having the same session key (Lines 65 to 66). Otherwise, we sample a uniformly random key as sID's session key (Line 67). In our protocol, if a session sID has computed a session key, then its context value ctxt can also uniquely indicates this session, and so we can use $\mathtt{SK}'[\mathsf{ctxt}]$ to store the session key of sID.
- To make the simulation consistent, in H, if $\mathcal{A}$'s RO query $(\mathsf{ctxt}, \mathsf{K}_1, \mathsf{K}_2, \widetilde{\mathsf{K}})$ corresponds to a session (i.e., $\mathtt{SK}'[\mathsf{ctxt}] \neq \bot$) and $\mathsf{K}_1, \mathsf{K}_2$, and $\widetilde{\mathsf{K}}$ are the three secret KEM keys in this session, then we return the session key of this session (i.e., $\mathtt{SK}'[\mathsf{ctxt}]$) as the RO response (Lines 21 to 24).

$\mathcal{A}$ cannot detect these modification unless it queries $\mathsf{G}(\mathsf{k}, IV)$ before IV are generated, or it queries $\mathsf{H}(\mathsf{ctxt}, \mathsf{K}_1, \mathsf{K}_2, \widetilde{\mathsf{K}})$ before ctxt is generated. Since such IV and ctxt are honestly generated in $\textsc{Session}_{\mathrm{I}}$ and $\textsc{Session}_{\mathrm{R}}$, respectively, by a union bound and public key entropy and ciphertext entropy of KEM_1 and KEM_0, we have

$$\left|\Pr\left[\mathbf{G}^{\mathcal{A}}_{0,b} \Rightarrow 1\right] - \Pr\left[\mathbf{G}^{\mathcal{A}}_{1,b} \Rightarrow 1\right]\right| \leq Q_{\mathsf{H}} S \cdot \left(\frac{1}{2^{\mu_0}} + \frac{1}{2^{\gamma_0}} + \frac{1}{2^{\gamma_1}}\right) + \frac{Q_{\mathsf{G}} S}{2^{\kappa}}.$$

Game $\mathbf{G}_{2,b}$: If $\mathcal{A}$ queried G on (k_i, IV) where party i is uncorrupted or the session sID corresponding to IV is unrevealed, then we raise flag $\mathsf{bad}_{\mathsf{st}}$ and abort the game (Lines 14 to 17). If $\mathsf{bad}_{\mathsf{st}}$ is not raised, $\mathcal{A}$'s view in $\mathbf{G}_{1,b}$ is the same as its view in $\mathbf{G}_{2,b}$. Since k_i and IV are generated by independent uniformly sampling, if party i is uncorrupted (resp., sID is unrevealed), then k_i (resp., IV) is uniform random in

Oracle $\mathsf{G}(\mathsf{k}, IV)$

01 **if** $\exists y$ s.t. $(\mathsf{k}, IV, y) \in \mathcal{L}_{\mathsf{G}}$: **return** y
02 $y \xleftarrow{\$} \{0,1\}^d$
03 **if** $\exists i$ s.t. $\mathsf{k} = \mathsf{k}_i$ $\wedge \mathtt{St}'_i[IV] \neq \perp$: // $\mathbf{G}_{1,b}$-$\mathbf{G}_{4,b}$
04 $(\varphi, \mathsf{st}') := \mathtt{St}'_i[IV]$ // $\mathbf{G}_{1,b}$-$\mathbf{G}_{4,b}$
05 $y := \varphi \oplus \mathsf{st}'$ // $\mathbf{G}_{1,b}$-$\mathbf{G}_{4,b}$
06 $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}, \mathsf{ct}_j, \mathsf{K}_j) := \mathsf{st}'$ // $\mathbf{G}_{3,b}$-$\mathbf{G}_{4,b}$
07 $\mathcal{C}_j := \mathcal{C}_j \backslash \{\mathsf{ct}_j\}$ // $\mathbf{G}_{3,b}$-$\mathbf{G}_{4,b}$
08 $\mathtt{Cor}'[\widetilde{\mathsf{pk}}] := 1$ // $\mathbf{G}_{4,b}$
09 $\mathcal{L}_{\mathsf{G}} := \mathcal{L}_{\mathsf{G}} \cup \{(\mathsf{k}, IV, y)\}$
10 **return** y

Oracle RevSt(sID)

11 **if** $\mathtt{Type}[\mathsf{sID}] \neq$ "In": **return** $\perp$
12 $i := \mathtt{Init}[\mathsf{sID}]$
13 $(IV, \varphi) := \mathtt{St}[\mathsf{sID}]$
14 **if** $\exists y$ s.t. $(\mathsf{k}_i, IV, y) \in \mathcal{L}_{\mathsf{G}}$
15 **if** $\neg\mathtt{Cor}[i]$ $\vee \neg\mathtt{revST}[\mathsf{sID}]$ // $\mathbf{G}_{2,b}$-$\mathbf{G}_{4,b}$
16 $\mathsf{bad}_{\mathsf{st}} := 1$ // $\mathbf{G}_{2,b}$-$\mathbf{G}_{4,b}$
17 **abort** // $\mathbf{G}_{2,b}$-$\mathbf{G}_{4,b}$
18 $\mathtt{revST}[\mathsf{sID}] := 1$
19 **return** $\mathtt{St}[\mathsf{sID}]$

Oracle $\mathsf{H}(\mathsf{ctxt}, \mathsf{K}_1, \mathsf{K}_2, \widetilde{\mathsf{K}})$

20 **if** $\exists \mathsf{SK}$ s.t. $(\mathsf{ctxt}, \mathsf{K}_1, \mathsf{K}_2, \widetilde{\mathsf{K}}, \mathsf{SK}) \in \mathcal{L}_{\mathsf{H}}$ **return** SK
21 **let** $(\mathsf{pk}_i, \mathsf{pk}_j, \widetilde{\mathsf{pk}}, \mathsf{ct}_1, \mathsf{ct}_2, \widetilde{\mathsf{ct}}) := \mathsf{ctxt}$ // $\mathbf{G}_{1,b}$-$\mathbf{G}_{4,b}$
22 **if** $\mathtt{SK}'[\mathsf{ctxt}] \neq \perp \wedge \mathsf{Decaps}_1(\mathsf{sk}_i, \mathsf{ct}_1) = \mathsf{K}_1 \wedge \mathsf{Decaps}_1(\mathsf{sk}_j, \mathsf{ct}_2) = \mathsf{K}_2$ // $\mathbf{G}_{1,b}$-$\mathbf{G}_{4,b}$
23 **if** $(\exists \widetilde{\mathsf{sk}}$ s.t. $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}) \in \widetilde{\mathcal{K}} \wedge \mathsf{Decaps}_0(\widetilde{\mathsf{sk}}, \widetilde{\mathsf{ct}}) = \widetilde{\mathsf{K}}) \vee (\widetilde{\mathsf{ct}}, \widetilde{\mathsf{K}}) \in \widetilde{\mathcal{L}}$ // $\mathbf{G}_{1,b}$-$\mathbf{G}_{4,b}$
24 $\mathsf{SK} := \mathtt{SK}'[\mathsf{ctxt}]$ // $\mathbf{G}_{1,b}$-$\mathbf{G}_{4,b}$
25 **if** $\big(\neg\mathtt{Cor}[i] \wedge \mathsf{ct}_1 \in \mathcal{C}_i \wedge \mathsf{K}_1 = \mathsf{Decap}_1(\mathsf{sk}'_i, \mathsf{ct}_1)\big)$ $\vee \big(\neg\mathtt{Cor}[j] \wedge \mathsf{ct}_2 \in \mathcal{C}_j \wedge \mathsf{K}_2 = \mathsf{Decap}_1(\mathsf{sk}'_j, \mathsf{ct}_2)\big)$ // $\mathbf{G}_{3,b}$-$\mathbf{G}_{4,b}$
26 $\mathsf{bad}_1 := 1$, **abort** // $\mathbf{G}_{3,b}$-$\mathbf{G}_{4,b}$
27 **if** $\exists \widetilde{\mathsf{sk}}$ s.t.$(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}) \in \widetilde{\mathcal{K}} \wedge (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{ct}}) \in \widetilde{\mathcal{C}} \wedge \mathsf{Decaps}_0(\widetilde{\mathsf{sk}}, \widetilde{\mathsf{ct}}) = \widetilde{\mathsf{K}} \wedge \neg\mathtt{Cor}[\widetilde{\mathsf{pk}}]$: // $\mathbf{G}_{4,b}$
28 $\mathsf{bad}_0 := 1$, **abort** // $\mathbf{G}_{4,b}$
29 $\mathsf{SK} \xleftarrow{\$} \mathcal{SK}$
30 $\mathcal{L}_{\mathsf{H}} := \mathcal{L}_{\mathsf{H}} \cup \{(\mathsf{ctxt}, \mathsf{K}_1, \mathsf{K}_2, \widetilde{\mathsf{K}}, \mathsf{SK})\}$
31 **return** SK

Figure 19: Oracles G, H, and RevSt of games $\mathbf{G}_{0,b}$-$\mathbf{G}_{4,b}(b \in \{0,1\})$ in Figure 18.

$\mathcal{A}$'s view. So, we have

$$\left|\Pr\left[\mathbf{G}^{\mathcal{A}}_{1,b} \Rightarrow 1\right] - \Pr\left[\mathbf{G}^{\mathcal{A}}_{2,b} \Rightarrow 1\right]\right| \leq \frac{NSQ_{\mathsf{G}}}{2^{\kappa}}.$$

Game $\mathbf{G}_{3,b}$: This game is the same as $\mathbf{G}_{2,b}$, except that we maintain lists $\{\mathcal{C}_i\}_{i\in[N]}$ (Line 08) to record ciphertexts generated in SessionI or SessionR, and use a flag bad_1 (Line 03) to determine whether the adversary queried H on KEM keys of ciphertexts in $\mathcal{C}_i$ and party $i(\in [N])$ is uncorrupted. If bad_1 is set as 1, then the game aborts (Lines 25 to 26). Concretely, the game simulator maintains lists $\mathcal{C}_i$ and flag bad_1 as follows:

- In SessionI, after generating ct_j, we record ct_j in $\mathcal{C}_j$ (Lines 19 to 20). Similarly, in SessionR, ct_i is recored in $\mathcal{C}_i$ (Line 43).
- If $\mathcal{A}$ queries $\mathsf{H}(\mathsf{pk}_i, \mathsf{pk}_j, \widetilde{\mathsf{pk}}, \mathsf{ct}_1, \mathsf{ct}_2, \widetilde{\mathsf{ct}}, \mathsf{K}_1, \mathsf{K}_2, \widetilde{\mathsf{K}})$, where ct_1 (resp., ct_2) belongs to $\mathcal{C}_i$ (resp., $\mathcal{C}_j$) and K_1 (resp., K_2) is the KEM key of ct_1 (resp., ct_2), and party i (resp., party j) is uncorrupted, then bad_1 is set as 1 and the game aborts.
- If $\mathcal{A}$ queries $\mathsf{G}(\mathsf{k}_i, IV)$, where this query corresponds to some session state which contains ct_j, then we delete ct_j from $\mathcal{C}_j$ (Lines 06 to 07). By the abort event $\mathsf{bad}_{\mathsf{st}}$ defined in $\mathbf{G}_{2,b}$, $\mathcal{A}$ issues this query only if party i is corrupted and the session state of the session corresponds to IV is revealed.

Intuitively, for some $i \in [N]$, if $\mathsf{ct} \in \mathcal{C}_i$ (which means the key of ct respect to pk_i is unrevealed) and party i is uncorrupted, then finding $\mathsf{K} = \mathsf{Decaps}_1(\mathsf{sk}_i, \mathsf{ct})$ means winning the game $\mathsf{OW\text{-}ChCCA}_{\mathsf{KEM}_1}$ (defined in Figure 3). For readability, we formalize it as Lemma E.1 and postpone its proof in Appendix E.1.

Lemma E.1 *With the notation and assumptions from the proof of Theorem 4.3, there is a PPT algorithm $\mathcal{B}_1$ with $\mathbf{T}(\mathcal{B}_1) \approx \mathbf{T}(\mathcal{A})$ and*

$$\left|\Pr\left[\boldsymbol{G}^{\mathcal{A}}_{2,b} \Rightarrow 1\right] - \Pr\left[\boldsymbol{G}^{\mathcal{A}}_{3,b} \Rightarrow 1\right]\right| \leq \mathsf{Adv}^{\mathsf{OW\text{-}ChCCA}}_{\mathcal{B}_1,\mathsf{KEM}_1}(\lambda) + \delta_1.$$

Game $\mathbf{G}_{4,b}$: We use list $\widetilde{\mathcal{C}}$ to record KEM_0 ciphertext $\widetilde{\mathsf{ct}}$ in $\textsc{Session}_\mathrm{R}$ if $\widetilde{\mathsf{ct}}$ is generated using some $\widetilde{\mathsf{pk}}$ that $(\widetilde{\mathsf{pk}}, \cdot) \in \widetilde{\mathcal{K}}$ (i.e., $\exists \widetilde{\mathsf{sk}}$ such that $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}) \in \widetilde{\mathcal{K}}$), see Lines 44 to 45. We also use list $\mathtt{Cor}'$ to determine if $\mathcal{A}$ revealed some KEM_0 key pair $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}})$ generated in $\textsc{Session}_\mathrm{I}$ (Line 08). Moreover, we use a flag bad_0 (Line 04) to determine whether the adversary queried H on KEM keys of ciphertexts of unrevealed KEM_0 public keys. If bad_0 is set as 1, then the game aborts (see Lines 27 to 28). Concretely, the game maintains lists $\mathtt{Cor}'$, $\widetilde{\mathcal{C}}$, and flag bad_0 as follows:

- In $\textsc{Session}_\mathrm{I}$, after generating $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}})$, the simulator sets $\mathtt{Cor}'[\widetilde{\mathsf{pk}}]$ as 0 to indicate that $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}})$ is not revealed yet (Line 24).
- If $\mathcal{A}$ queries G on (k_i, IV) and $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}})$ is the key pair of $\mathtt{St}'_i[IV]$ (Line 06), then $\mathtt{Cor}'[\widetilde{\mathsf{pk}}]$ is set as 1 to indicate that $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}})$ is revealed to $\mathcal{A}$ (Line 08). By the abort event $\mathsf{bad}_{\mathsf{st}}$ defined in $\mathbf{G}_{2,b}$, $\mathcal{A}$ issues this query only if party i is corrupted and the session state of the session corresponds to IV is revealed.
- In $\textsc{Session}_\mathrm{R}$, after generating $\widetilde{\mathsf{ct}}$ using $\widetilde{\mathsf{pk}}$, if $\widetilde{\mathsf{pk}}$ is generated by $\textsc{Session}_\mathrm{I}$ ($\exists \widetilde{\mathsf{sk}}$ s.t. $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}) \in \widetilde{\mathcal{K}}$), then $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{ct}})$ will be recorded in $\widetilde{\mathcal{C}}$ (Lines 44 to 45).
- If $\mathcal{A}$ queries $\mathsf{H}(\mathsf{pk}_i, \mathsf{pk}_j, \widetilde{\mathsf{pk}}, \mathsf{ct}_1, \mathsf{ct}_2, \widetilde{\mathsf{ct}}, \mathsf{K}_1, \mathsf{K}_2, \widetilde{\mathsf{K}})$, where $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{ct}})$ belongs to $\widetilde{\mathcal{C}}$, $\widetilde{\mathsf{K}}$ is the KEM key of $\widetilde{\mathsf{ct}}$ wrt $\widetilde{\mathsf{pk}}$, and the secret key of $\widetilde{\mathsf{pk}}$ is not revealed (i.e., $\mathtt{Cor}'[\widetilde{\mathsf{pk}}] = 0$), then bad_0 is set as 1, and the game aborts.

Intuitively, if $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}) \in \widetilde{\mathcal{K}}$, $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{ct}}) \in \widetilde{\mathcal{C}}$, and $\mathtt{Cor}'[\widetilde{\mathsf{pk}}] = 0$, then finding $\widetilde{\mathsf{K}} = \mathsf{Decaps}_0(\widetilde{\mathsf{sk}}, \mathsf{ct})$ means winning the game $\mathsf{OW\text{-}ChCCA}_{\mathsf{KEM}_0}$ (defined in Figure 3). Actually, we have the following lemma, which will be proven in Appendix E.2.

Lemma E.2 *With the notation and assumptions from the proof of Theorem 4.3, there is a PPT algorithm $\mathcal{B}_0$ with $\mathbf{T}(\mathcal{B}_0) \approx \mathbf{T}(\mathcal{A})$ and*

$$\left|\Pr\left[\boldsymbol{G}^{\mathcal{A}}_{3,b} \Rightarrow 1\right] - \Pr\left[\boldsymbol{G}^{\mathcal{A}}_{4,b} \Rightarrow 1\right]\right| \leq \mathsf{Adv}^{\mathsf{OW\text{-}ChCCA}}_{\mathcal{B}_0,\mathsf{KEM}_0}(\lambda) + \delta_0.$$

Here we bound $|\Pr[\mathbf{G}^{\mathcal{A}}_{4,0} \Rightarrow 1] - \Pr[\mathbf{G}^{\mathcal{A}}_{4,1} \Rightarrow 1]|$. The adversary distinguishes these two games if it queries the hash input of $\mathtt{SK}[\mathsf{sID}^*]$, where sID^* can be arbitrary valid and fresh test session. Let sID^* be an arbitrary session belongs to $\mathcal{S}$. By the validity definition (Figure 17), sID^* must fall into the attacking types listed in Table 1. Let $(\mathsf{pk}_i, \mathsf{pk}_j, \widetilde{\mathsf{pk}}, \mathsf{ct}_1, \mathsf{ct}_2, \widetilde{\mathsf{ct}}, \mathsf{K}_1, \mathsf{K}_2, \widetilde{\mathsf{K}})$ be the hash input of $\mathtt{SK}[\mathsf{sID}^*]$, where $\mathsf{K}_1 = \mathsf{Decaps}_1(\mathsf{sk}_i, \mathsf{ct}_1)$, $\mathsf{K}_2 = \mathsf{Decaps}_1(\mathsf{sk}_j, \mathsf{ct}_2)$, and $\widetilde{\mathsf{K}} = \mathsf{Decaps}_0(\widetilde{\mathsf{sk}}, \widetilde{\mathsf{ct}})$. We consider all attack types listed in Table 1:

- $\textsc{Attack}$ (1), (3). If sID^* is type (1), by definition, $\mathcal{A}$ never reveals the state of sID^* and thus the state encryption vector of this session (denoted as IV^*) is not revealed. Since sID^* has a matching session, there exists $\widetilde{\mathsf{sk}}$ such that $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}) \in \widetilde{\mathcal{K}}$ and $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{ct}}) \in \widetilde{\mathcal{C}}$. By the abort event $\mathsf{bad}_{\mathsf{st}}$ (Line 02), $\mathcal{A}$ cannot query G on (k_{i^*}, IV^*), and the code in Line 08 will never be executed, i.e., $\mathtt{Cor}'[\widetilde{\mathsf{pk}}]$ is always 0. Therefore, by the abort event bad_0 (Lines 27 to 28), $\mathcal{A}$ cannot query $\widetilde{\mathsf{K}}$. Similarly, the above argument still applies when sID^* is type (3), $\mathcal{A}$ cannot query $\widetilde{\mathsf{K}}$ in these cases.
- $\textsc{Attack}$ (2), (4). In these cases, party i^* is uncorrupted, and by definition (sID^* has (partially) matching session), we have $\mathsf{ct}_1 \in \mathcal{C}_{i^*}$. By the abort event bad_1 (Lines 25 to 26), $\mathcal{A}$ cannot query K_1 in this case.
- $\textsc{Attack}$ (5). In this case, by definition, both parties i^* and j^* are uncorrupted and $\mathsf{ct}_2 \in \mathcal{C}_{j^*}$. Since i^* is uncorrupted, the abort event bad_0 assures that $\mathcal{A}$ cannot query $\mathsf{G}(\mathsf{k}_i, IV)$ to trigger the codes in Lines 06 to 07, and thus ct_2 will never be deleted from $\mathcal{C}_{j^*}$. By the abort event bad_1 (Lines 25 to 26), $\mathcal{A}$ cannot query K_2 in this case.
- $\textsc{Attack}$ (6). In this case, by definition, party j^* is uncorrupted and $\mathsf{ct}_2 \in \mathcal{C}_{j^*}$, and the state of sID^* is not revealed. This means that $\mathcal{A}$ cannot query $\mathsf{G}(\mathsf{k}_i, IV)$ to trigger the codes in Lines 06 to 07 by the abort event bad_0, and thus ct_2 will never be deleted from $\mathcal{C}_{j^*}$. By the abort event bad_1 (Lines 25 to 26), $\mathcal{A}$ cannot query K_2 in this case.
- $\textsc{Attack}$ (7). In this case, by definition, party i^* is uncorrupted and $\mathsf{ct}_1 \in \mathcal{C}_{i^*}$. By the abort event bad_1 (Lines 25 to 26), $\mathcal{A}$ cannot query K_1 in this case.

Therefore, for any fresh and valid $\mathsf{sID}^* \in \mathcal{S}$, the adversary cannot query the hash input of $\mathtt{SK}[\mathsf{sID}^*]$. Since H is a random oracle, the adversary cannot distinguish $\mathtt{SK}[\mathsf{sID}^*]$ from a random key, and thus in

TEST oracle, SK_0 has the same distribution as SK_1. We have

$$\Pr\left[\mathbf{G}^{\mathcal{A}}_{4,0} \Rightarrow 1\right] = \Pr\left[\mathbf{G}^{\mathcal{A}}_{4,1} \Rightarrow 1\right]$$

By combining all probabilistic difference and Lemmata E.1 and E.2, we have

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{wFS\text{-}St}}_{\mathcal{A},\mathsf{AKE}}(\lambda) &= \left|\Pr\left[\mathsf{wFS\text{-}St}^{\mathcal{A}}_{0,\mathsf{AKE}} \Rightarrow 1\right] - \Pr\left[\mathsf{wFS\text{-}St}^{\mathcal{A}}_{1,\mathsf{AKE}} \Rightarrow 1\right]\right| \\
&\leq 2\mathsf{Adv}^{\mathsf{OW\text{-}ChCCA}}_{\mathcal{B}_1,\mathsf{KEM}_1}(\lambda) + 2\mathsf{Adv}^{\mathsf{OW\text{-}ChCCA}}_{\mathcal{B}_0,\mathsf{KEM}_0}(\lambda) \\
&+ 2\delta_1 + 2\delta_0 + \frac{N^2}{2^{\mu_1-1}} + \frac{(N+1)SQ_{\mathsf{G}}}{2^{\kappa-1}} + \frac{2(Q^2_{\mathsf{H}} + S^2)}{|\mathcal{SK}|} \\
&+ 2S\cdot(Q_{\mathsf{H}} + S)\cdot\left(\frac{1}{2^{\gamma_1}} + \frac{1}{2^{\gamma_0}} + \frac{1}{2^{\mu_0}}\right) + \frac{Q^2_{\mathsf{G}} + N^2 + S^2}{2^{d-1}},
\end{aligned}$$

as stated in Theorem 4.3. □

E.1 Proof of Lemma E.1

Lemma E.1. Let $\Pr[\mathsf{bad}_1]$ be the probability that flag bad_1 is set as 1. By the argument in the proof of Theorem 4.3, we have

$$\left|\Pr\left[\mathbf{G}^{\mathcal{A}}_{2,b} \Rightarrow 1\right] - \Pr\left[\mathbf{G}^{\mathcal{A}}_{3,b} \Rightarrow 1\right]\right| \leq \Pr[\mathsf{bad}_1]$$

We construct an adversary $\mathcal{B}_1$ that simulates $\mathbf{G}_{3,b}$ for $\mathcal{A}$ to break OW-ChCCA security of KEM_1 in protocol AKE. If $\mathcal{A}$ triggers bad_1, then $\mathcal{B}_1$ breaks KEM_1. The construction of $\mathcal{B}_1$ is shown in Figures 20 and 21.

By the definition of OW-ChCCA security game (Figure 3), $\mathcal{B}_1$ has access to $\textsc{Enc}_1, \textsc{Dec}_1, \textsc{Corr}_1$, and $\textsc{Check}_1$. $\mathcal{B}_1$ uses (i^*, ct^*) to record one-way solution of the OW-ChCCA game. For each party $i \in [N]$, $\mathcal{B}_1$ sets pk_i (from its input) as the public key of party i, and generates party i's state encryption key k_i. Since now $\mathcal{B}_1$ does not have secret key of pk_i, it uses oracles $\textsc{Dec}_1, \textsc{Corr}_1$, and $\textsc{Check}_1$ to simulate $\mathbf{G}_{3,b}$. Specifically,

- In $\textsc{Session}_{\mathrm{I}}$ (resp., $\textsc{Session}_{\mathrm{R}}$), $\mathcal{B}_1$ generates ct_j (resp., ct_i) by querying $\textsc{Enc}_1(j)$ (resp., $\textsc{Enc}_1(i)$). Since $\mathcal{B}_1$ does not have the KEM_1 key K_j of ct_j, $\mathcal{B}_1$ leaves it unknown until $\mathcal{A}$ queries G on (k_i, IV). When $\mathcal{A}$ queries $\mathsf{G}(\mathsf{k}_i, IV)$, $\mathcal{B}_1$ queries $\textsc{Reveal}_1(j, \mathsf{ct}_j)$ and gets K_j (Lines 05 to 06).
- In $\textsc{Session}_{\mathrm{R}}$, if $\mathsf{ct}_j \notin \mathcal{C}_j$ (i.e., the received ct_j is not generated in $\textsc{Session}_{\mathrm{I}}$), then $\mathcal{B}_1$ uses $\textsc{Dec}_1$ to decrypt the received KEM_1 ciphertext ct_j to determine if ct_j is valid. If ct_j is valid, then $\mathcal{B}_1$ leaves the KEM key of ct_j as unknown, and uses $\textsc{Check}_0$ to determine if $\mathcal{A}$'s queries to H include ct_j's KEM key. If $\mathsf{ct}_j \in \mathcal{C}_j$, then ct_j is a OW-ChCCA challenge ciphertext wrt pk_i. In this case, the simulator leaves the KEM key of ct_j as unknown and uses $\textsc{Check}_0$ to search $\mathcal{A}$'s queries to H and recover the KEM_1 key from them to break the OW-ChCCA security of KEM_1. In $\textsc{Der}_{\mathrm{I}}$, we use the similar strategy described above to deal with the received KEM_1 ciphertext ct_i.
- In H, $\mathcal{B}_1$ uses $\textsc{Check}_1$ to determine if $\mathsf{Decaps}_1(\mathsf{sk}_i, \mathsf{ct}) = \mathsf{K}$. If $\mathcal{A}$ triggers bad_1 event, then $\mathcal{B}_1$ outputs the corresponding identity, challenge ciphertext, and the KEM key, and aborts the simulation (see Lines 24 to 27).
- If $\mathcal{A}$ queries $\mathsf{G}(\mathsf{k}_i, IV)$, where this query corresponds to some session state which contains ct_j, then $\mathcal{B}_1$ deletes ct_j from $\mathcal{C}_j$ (Lines 06 to 07), queries $\textsc{Reveal}_1(j, \mathsf{ct}_j)$ to get K_j so that it can simulate G in a consistent way.
- When $\mathcal{A}$ corrupts party i, $\mathcal{B}_1$ queries $\textsc{Corr}_1(i)$ and returns the responded secret key.

Let $(i^*, \mathsf{ct}^*, \mathsf{K})$ be $\mathcal{B}_1$'s final output. If $\mathcal{A}$ triggers bad_1 event, then by definition, we have $\mathtt{Cor}[i^*] = 0$, $\mathsf{ct}^* \in \mathcal{C}_{i^*}$, and K is the KEM_1 key of ct^* wrt pk_{i^*}. $\mathtt{Cor}[i^*] = 0$ means that party i^* is uncorrupted, and $\mathcal{B}_1$ never queries $\textsc{Corr}_1(i^*)$. $\mathsf{ct}^* \in \mathcal{C}_{i^*}$ means that ct^* is a challenge ciphertext wrt pk_{i^*} and $\mathcal{B}_1$ never queries $\textsc{Reveal}_1(i^*, \mathsf{ct}^*)$. Therefore, $(i^*, \mathsf{ct}^*, \mathsf{K})$ is a valid solution of game $\mathsf{OW\text{-}ChCCA}_{\mathsf{KEM}_1}(\lambda)$ (Figure 3). We also need to count the error term δ_1 into the final bound since KEM_1 is $(1-\delta_1)$-correct. That is, we have

$$\left|\Pr\left[\mathbf{G}^{\mathcal{A}}_{2,b} \Rightarrow 1\right] - \Pr\left[\mathbf{G}^{\mathcal{A}}_{3,b} \Rightarrow 1\right]\right| \leq \Pr[\mathsf{bad}_1] \leq \mathsf{Adv}^{\mathsf{OW\text{-}ChCCA}}_{\mathcal{B}_1,\mathsf{KEM}_1}(\lambda) + \delta_1,$$

as stated in Lemma E.1.

□

$\mathcal{B}_1{}^{\mathrm{O}_{\mathsf{KEM}_1}}(\mathsf{par}, (\mathsf{pk}_i)_{i\in[N]})$

$i^* := 0, \mathsf{ct}^* := \bot$ //Record the solution
$\mathsf{cnt} := 0, \mathcal{S} := \emptyset, \mathsf{bad}_{\mathsf{st}} := 0, \mathsf{bad}_1 := 0$
$\widetilde{\mathsf{par}} \leftarrow \mathsf{Setup}_0(1^\lambda)$, $\mathsf{par}' := (\widetilde{\mathsf{par}}, \mathsf{par})$
for $i \in [N]$: $\mathsf{k}_i \leftarrow \{0,1\}^\kappa, \mathcal{C}_i := \emptyset$
$b' \leftarrow \mathcal{A}^{\mathsf{H},\mathsf{G},\mathrm{O}}(\mathsf{par}', (\mathsf{pk}_i)_{i\in[N]})$
return $(i^*, \mathsf{ct}^*, \bot)$

Oracle $\textsc{Der}_{\mathrm{I}}(\mathsf{sID}, \mathsf{M})$

if $\mathtt{Used}[\mathsf{sID}] = 1$: **return** $\bot$
if $\mathtt{St}[\mathsf{sID}] = \bot$: **return** $\bot$
if $\mathtt{SK}[\mathsf{sID}] \neq \bot$: **return** $\bot$
$\mathtt{Used}[\mathsf{sID}] := 1$
$(i, j) := (\mathtt{Init}[\mathsf{sID}], \mathtt{Resp}[\mathsf{sID}])$
let $(IV, \varphi) := \mathtt{St}[\mathsf{sID}], (\varphi, \mathsf{st}') := \mathtt{St}'_i[IV]$
let $(\widetilde{\mathsf{ct}}, \mathsf{ct}_i) := \mathsf{M}, (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}, \mathsf{ct}_j, \mathsf{K}_j) := \mathsf{st}'$
if $\mathsf{ct}_i \notin \mathcal{C}_i$:
  $\mathsf{K}_i := \textsc{Dec}_1(i, \mathsf{ct}_i)$
  if $\mathsf{K}_i = \bot$: **return** $\bot$
$\widetilde{\mathsf{K}} := \mathsf{Decaps}_0(\widetilde{\mathsf{sk}}, \widetilde{\mathsf{ct}})$
if $\widetilde{\mathsf{K}} = \bot$: **return** $\bot$
$\mathsf{ctxt} := (\mathsf{pk}'_i, \mathsf{pk}'_j, \widetilde{\mathsf{pk}}, \mathsf{ct}_i, \mathsf{ct}_j, \widetilde{\mathsf{ct}})$
if $\mathtt{SK}'[\mathsf{ctxt}] \neq \bot$: $\mathsf{SK} := \mathtt{SK}'[\mathsf{ctxt}]$
else $\mathtt{SK}'[\mathsf{ctxt}] := \mathsf{SK} \leftarrow \mathcal{SK}$
$(\mathtt{R}[\mathsf{sID}], \mathtt{SK}[\mathsf{sID}]) := (\mathsf{M}, \mathsf{SK})$
return 1

Oracle $\textsc{Cor}(i)$

$\mathtt{Cor}[i] := 1$, $\mathsf{sk}_i := \textsc{Corr}_1(i)$
return $(\mathsf{sk}_i, \mathsf{sk}_i)$

Oracle $\textsc{Session}_{\mathrm{I}}((i, j) \in [N]^2)$

$\mathsf{cnt} := \mathsf{cnt} + 1, \mathsf{sID} := \mathsf{cnt}$
$(\mathtt{Init}[\mathsf{sID}], \mathtt{Resp}[\mathsf{sID}]) := (i, j)$
$\mathtt{Type}[\mathsf{sID}] :=$ "In"
$\mathsf{ct}_j \leftarrow \textsc{Enc}_1(j)$, $\mathcal{C}_j := \mathcal{C}_j \cup \{\mathsf{ct}_j\}$
$(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}) \leftarrow \mathsf{KG}_0(\widetilde{\mathsf{par}})$
$\widetilde{\mathcal{K}} := \widetilde{\mathcal{K}} \cup \{(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}})\}$
$IV \leftarrow \{0,1\}^\kappa, \varphi \leftarrow \{0,1\}^d$
$\mathsf{st}' := (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}, \mathsf{ct}_j, \bot)$
$\mathtt{St}'_i[IV] := (\varphi, \mathsf{st}')$, $\mathsf{M}_i := (\widetilde{\mathsf{pk}}, \mathsf{ct}_j)$
$(\mathtt{I}[\mathsf{sID}], \mathtt{St}[\mathsf{sID}]) := (\mathsf{M}_i, \mathsf{st})$
return $(\mathsf{sID}, \mathsf{M}_i)$

Oracle $\textsc{Session}_{\mathrm{R}}((i, j) \in [N]^2, \mathsf{M})$

$\mathsf{cnt} := \mathsf{cnt} + 1, \mathsf{sID} := \mathsf{cnt}$
$(\mathtt{Init}[\mathsf{sID}], \mathtt{Resp}[\mathsf{sID}]) := (i, j)$
$\mathtt{Type}[\mathsf{sID}] :=$ "Re"
let $(\widetilde{\mathsf{pk}}, \mathsf{ct}_j) := \mathsf{M}$
if $\mathsf{ct}_j \notin \mathcal{C}_j$
  $\mathsf{K}_j := \textsc{Dec}_1(j, \mathsf{ct}_j)$
  if $\mathsf{K}_j = \bot$: **return** $\bot$
$(\widetilde{\mathsf{ct}}, \widetilde{\mathsf{K}}) \leftarrow \mathsf{Encaps}_0(\widetilde{\mathsf{pk}})$
$\mathsf{ct}_i \leftarrow \textsc{Enc}_1(i)$
if $(\widetilde{\mathsf{pk}}, \cdot) \notin \widetilde{\mathcal{K}}$: $\widetilde{\mathcal{L}} := \widetilde{\mathcal{L}} \cup \{(\widetilde{\mathsf{ct}}, \widetilde{\mathsf{K}})\}$
$\mathcal{C}_i := \mathcal{C}_i \cup \{\mathsf{ct}_i\}, \mathsf{SK} \leftarrow \mathcal{SK}$
$\mathsf{ctxt} := (\mathsf{pk}'_i, \mathsf{pk}'_j, \widetilde{\mathsf{pk}}, \mathsf{ct}_i, \mathsf{ct}_j, \widetilde{\mathsf{ct}})$
$\mathtt{SK}'[\mathsf{ctxt}] := \mathsf{SK}$, $\mathtt{SK}[\mathsf{sID}] := \mathsf{SK}$
$\mathsf{M}_j := (\widetilde{\mathsf{ct}}, \mathsf{ct}_i)$
$(\mathtt{I}[\mathsf{sID}], \mathtt{R}[\mathsf{sID}]) := (\mathsf{M}, \mathsf{M}_j)$
return $(\mathsf{sID}, \mathsf{M}_j)$

Figure 20: $\mathcal{B}_1$ has access to $\mathrm{O}_{\mathsf{KEM}_1} := \{\textsc{Enc}_1, \textsc{Dec}_1, \textsc{Reveal}_1, \textsc{Corr}_1, \textsc{Check}_1\}$. $\mathcal{A}$ has access to G, H, and $\mathrm{O} := (\textsc{Session}_{\mathrm{I}}, \textsc{Der}_{\mathrm{I}}, \textsc{Session}_{\mathrm{R}}, \textsc{RevK}, \textsc{Cor}, \textsc{Test})$. Oracle $\textsc{Test}$ is the same as $\mathbf{G}_{3,b}$ in Figure 18. Oracles $\mathsf{G}, \mathsf{H}, \textsc{Cor}$, and $\textsc{RevSt}$ are given in Figure 21. $\mathcal{B}_1$ uses the oracles (highlighted) provided by $\mathsf{OW\text{-}ChCCA}_{\mathsf{KEM}}$ to simulate $\mathbf{G}_{3,b}$.

Oracle $\mathsf{G}(\mathsf{k}, IV)$

01 **if** $\exists y$ s.t. $(\mathsf{k}, IV, y) \in \mathcal{L}_{\mathsf{G}}$: **return** y
02 $y \leftarrow \{0,1\}^d$
03 **if** $\exists i$ s.t. $\mathsf{k} = \mathsf{k}_i \wedge \mathtt{St}'_i[IV] \neq \perp$
04 $(\varphi, \mathsf{st}') := \mathtt{St}'_i[IV]$
05 $\mathsf{st}' := (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}, \mathsf{ct}_j, \perp)$
06 $\mathsf{K}_j := \textsc{Reveal}_1(j, \mathsf{ct}_j)$
07 $\mathcal{C}_j := \mathcal{C}_j \backslash \{\mathsf{ct}_j\}$
08 $y := \varphi \oplus (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}, \mathsf{ct}_j, \mathsf{K}_j)$
09 $\mathcal{L}_{\mathsf{G}} := \mathcal{L}_{\mathsf{G}} \cup \{(\mathsf{k}, IV, y)\}$
10 **return** y

Oracle $\textsc{RevSt}(\mathsf{sID})$

11 **if** $\mathtt{Type}[\mathsf{sID}] \neq$ "In": **return** $\perp$
12 $i := \mathtt{Init}[\mathsf{sID}], (IV, \varphi) := \mathtt{St}[\mathsf{sID}]$
13 **if** $\exists y$ s.t. $(\mathsf{k}_i, IV, y) \in \mathcal{L}_{\mathsf{G}}$
14 **if** $\neg\mathtt{Cor}[i] \vee \neg\mathtt{revST}[\mathsf{sID}]$
15 $\mathsf{bad}_{\mathsf{st}} := 1$
16 **output** $(i^*, \mathsf{ct}^*, \perp)$ and **abort**
17 $\mathtt{revST}[\mathsf{sID}] := 1$
18 **return** $\mathtt{St}[\mathsf{sID}]$

Oracle $\mathsf{H}(\mathsf{ctxt}, \mathsf{K}_1, \mathsf{K}_2, \widetilde{\mathsf{K}})$

19 **if** $\exists \mathsf{SK}$ s.t. $(\mathsf{ctxt}, \mathsf{K}_1, \mathsf{K}_2, \widetilde{\mathsf{K}}, \mathsf{SK}) \in \mathcal{L}_{\mathsf{H}}$ **return** SK
20 **let** $(\mathsf{pk}_i, \mathsf{pk}_j, \widetilde{\mathsf{pk}}, \mathsf{ct}_1, \mathsf{ct}_2, \widetilde{\mathsf{ct}}) := \mathsf{ctxt}$
21 **if** $\mathtt{SK}'[\mathsf{ctxt}] \neq \perp \wedge \textsc{Check}_1(i, \mathsf{ct}_1, \mathsf{K}_1) = 1 \wedge \textsc{Check}_1(j, \mathsf{ct}_2, \mathsf{K}_2) = 1$
22 **if** $(\exists \widetilde{\mathsf{sk}}$ s.t. $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}) \in \widetilde{\mathcal{K}} \wedge \mathsf{Decaps}_0(\widetilde{\mathsf{sk}}, \widetilde{\mathsf{ct}}) = \widetilde{\mathsf{K}}) \vee (\widetilde{\mathsf{ct}}, \widetilde{\mathsf{K}}) \in \widetilde{\mathcal{L}}$
23 $\mathsf{SK} := \mathtt{SK}'[\mathsf{ctxt}]$
24 **if** $\neg\mathtt{Cor}[i] \wedge \mathsf{ct}_1 \in \mathcal{C}_i \wedge \textsc{Check}_1(i, \mathsf{ct}_1, \mathsf{K}_1) = 1$
25 $\mathsf{bad}_1 := 1$, $i^* := i, \mathsf{ct}^* := \mathsf{ct}_1$, **output** $(i^*, \mathsf{ct}^*, \mathsf{K}_1)$ and **abort**
26 **if** $\neg\mathtt{Cor}[j] \wedge \mathsf{ct}_2 \in \mathcal{C}_j \wedge \textsc{Check}_1(j, \mathsf{ct}_2, \mathsf{K}_2) = 1$
27 $\mathsf{bad}_1 := 1$, $i^* := j, \mathsf{ct}^* := \mathsf{ct}_2$, **output** $(i^*, \mathsf{ct}^*, \mathsf{K}_2)$ and **abort**
28 $\mathsf{SK} \leftarrow \mathcal{SK}$
29 $\mathcal{L}_{\mathsf{H}} := \mathcal{L}_{\mathsf{H}} \cup \{(\mathsf{ctxt}, \mathsf{K}_1, \mathsf{K}_2, \widetilde{\mathsf{K}}, \mathsf{SK})\}$
30 **return** SK

Figure 21: Oracles $\mathsf{G}, \mathsf{H}, \textsc{Cor}$ and $\textsc{RevSt}$ in the proof of Lemma E.1. $\mathcal{B}_1$ uses the oracles (highlighted) provided by the game $\mathsf{OW\text{-}ChCCA}_{\mathsf{KEM}}(\lambda)$ to simulate $\mathbf{G}_{3,b}$.

E.2 Proof of Lemma E.2

Lemma E.2. In this section, we bound the probability difference between $\mathbf{G}_{3,b}$ with $\mathbf{G}_{4,b}$ in the proof of Lemma E.2. Let $\Pr[\mathsf{bad}_0]$ be the probability that flag bad_0 is set as 1 in $\mathbf{G}_{4,b}$. By the argument in the proof of Theorem 4.3,

$$\left|\Pr\left[\mathbf{G}^{\mathcal{A}}_{3,b} \Rightarrow 1\right] - \Pr\left[\mathbf{G}^{\mathcal{A}}_{4,b} \Rightarrow 1\right]\right| \leq \Pr[\mathsf{bad}_0] + \delta_0.$$

We construct an adversary $\mathcal{B}_0$ that simulates $\mathbf{G}_{4,b}$ for $\mathcal{A}$ to break $\mathsf{OW\text{-}ChCCA}$ security of KEM_0 in protocol AKE. If $\mathcal{A}$ triggers bad_0, then $\mathcal{B}_0$ breaks KEM_0. The construction of $\mathcal{B}_0$ is shown in Figures 22 and 23.

By the definition of $\mathsf{OW\text{-}ChCCA}$ security game (Figure 3), $\mathcal{B}_0$ has access to $\textsc{Enc}_0, \textsc{Dec}_0, \textsc{Corr}_0$, and $\textsc{Check}_0$. $\mathcal{B}_0$ uses $(t^*, \widetilde{\mathsf{ct}}^*)$ to record one-way solution of the $\mathsf{OW\text{-}ChCCA}$ game. $\mathcal{B}_0$'s input includes S (i.e., the maximal number of sessions) KEM_0 public keys, and it embeds these public keys into sessions. Specifically,

- In $\textsc{Session}_{\mathrm{I}}$ (with session identity sID), $\mathcal{B}_0$ sets $\widetilde{\mathsf{pk}} := \widetilde{\mathsf{pk}}^*_{\mathsf{sID}}$ and records $(\widetilde{\mathsf{pk}}, \mathsf{sID})$ in $\widetilde{\mathcal{K}}$. Since $\mathcal{B}_0$ does not have the private key $\widetilde{\mathsf{sk}}$ of $\widetilde{\mathsf{pk}}$, it needs to record sID so that it can query $\textsc{Dec}_0$ wrt $\widetilde{\mathsf{pk}}$, and it also leaves $\widetilde{\mathsf{sk}}$ unknown until $\mathcal{A}$ queries G on (k_i, IV). When $\mathcal{A}$ queries $\mathsf{G}(\mathsf{k}_i, IV)$, $\mathcal{B}_0$ queries $\textsc{Corr}_0(\mathsf{sID})$ and gets $\widetilde{\mathsf{sk}}$ (Lines 06 to 10).
- In $\textsc{Session}_{\mathrm{R}}$, if the received $\widetilde{\mathsf{pk}}$ is not generated in $\textsc{Session}_{\mathrm{I}}$, $\mathcal{B}_0$ generates $(\widetilde{\mathsf{ct}}, \widetilde{\mathsf{K}})$ using $\widetilde{\mathsf{pk}}$ normally. If $\widetilde{\mathsf{pk}}$ is generated in $\textsc{Session}_{\mathrm{I}}$ (i.e., $\exists t$ s.t. $(\widetilde{\mathsf{pk}}, t) \in \widetilde{\mathcal{K}}$), then $\mathcal{B}_0$ queries $\textsc{Enc}_0(t)$ to get a challenge ciphertext $\widetilde{\mathsf{ct}}$ and records it in $\widetilde{\mathcal{C}}$.
- In $\textsc{Der}_{\mathrm{I}}$, if the received $\widetilde{\mathsf{ct}}$ is not generated in $\textsc{Session}_{\mathrm{R}}$ (i.e., $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{ct}}) \notin \widetilde{\mathcal{C}}$), $\mathcal{B}_0$ finds t s.t. $(\widetilde{\mathsf{pk}}, t) \in \widetilde{\mathcal{K}}$ and queries $\textsc{Dec}_0(t, \widetilde{\mathsf{ct}})$ to decrypt $\widetilde{\mathsf{ct}}$ and determine if $\widetilde{\mathsf{ct}}$ is valid (Lines 18 to 20). If $\widetilde{\mathsf{ct}}$ is valid, then the simulator leaves the KEM key of $\widetilde{\mathsf{ct}}$ as unknown and uses $\textsc{Check}_0$ to determine if $\mathcal{A}$'s queries to H include $\widetilde{\mathsf{ct}}$'s KEM key. If the received $\widetilde{\mathsf{ct}}$ is generated in $\textsc{Session}_{\mathrm{R}}$ (i.e., $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{ct}}) \in \widetilde{\mathcal{C}}$), then this $\widetilde{\mathsf{ct}}$ is a $\mathsf{OW\text{-}ChCCA}$ challenge ciphertext wrt $\widetilde{\mathsf{pk}}$. In this case, the simulator also leaves the

Game $\mathcal{B}_0{}^{\mathrm{O}_{\mathsf{KEM}_0}}(\widetilde{\mathsf{par}}, (\widetilde{\mathsf{pk}}_t^*)_{t\in[S]})$

$t^* := 0, \widetilde{\mathsf{ct}}^* := \bot, \mathsf{cnt} := 0, \mathcal{S} := \emptyset$
$\mathsf{bad}_{\mathsf{st}} := 0, \mathsf{bad}_1 := 0, \mathsf{bad}_0 := 0$
$\mathsf{par} \leftarrow \mathsf{Setup}_1(1^\lambda), \mathsf{par}' := (\widetilde{\mathsf{par}}, \mathsf{par})$
for $i \in [N]$:
  $(\mathsf{pk}_i, (\mathsf{sk}_i, \mathsf{k}_i)) \leftarrow \mathsf{KG}'(\mathsf{par}')$
  $\mathcal{C}_i := \emptyset$
$b' \leftarrow \mathcal{A}^{\mathsf{H},\mathsf{G},\mathrm{O}}(\mathsf{par}', (\mathsf{pk}_i)_{i\in[N]})$
return $(i^*, \widetilde{\mathsf{ct}}^*, \bot)$

Oracle $\textsc{Der}_{\mathrm{I}}(\mathsf{sID}, (\widetilde{\mathsf{ct}}, \mathsf{ct}_i))$

if $\mathtt{Used}[\mathsf{sID}] = 1$: **return** $\bot$
if $\mathtt{St}[\mathsf{sID}] = \bot$: **return** $\bot$
if $\mathtt{SK}[\mathsf{sID}] \neq \bot$: **return** $\bot$
$\mathtt{Used}[\mathsf{sID}] := 1$
let $(i, j) := (\mathtt{Init}[\mathsf{sID}], \mathtt{Resp}[\mathsf{sID}])$
let $(IV, \varphi) := \mathtt{St}[\mathsf{sID}]$
let $(\varphi, (\widetilde{\mathsf{pk}}, \bot, \mathsf{ct}_j, \mathsf{K}_j)) := \mathtt{St}'_i[IV]$
$\mathsf{K}_i := \mathsf{Decaps}_1(\mathsf{sk}'_i, \mathsf{ct})$
if $\mathsf{K}_i = \bot$: **return** $\bot$
if $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{ct}}) \notin \widetilde{\mathcal{C}}$
  $\widetilde{\mathsf{K}} := \textsc{Dec}_0(\mathsf{sID}, \widetilde{\mathsf{ct}})$
  if $\widetilde{\mathsf{K}} = \bot$: **return** $\bot$
$\mathsf{ctxt} := (\mathsf{pk}'_i, \mathsf{pk}'_j, \widetilde{\mathsf{pk}}, \mathsf{ct}_i, \mathsf{ct}_j, \widetilde{\mathsf{ct}})$
if $\mathtt{SK}'[\mathsf{ctxt}] \neq \bot$: $\mathsf{SK} := \mathtt{SK}'[\mathsf{ctxt}]$
else $\mathtt{SK}'[\mathsf{ctxt}] := \mathsf{SK} \leftarrow \mathcal{SK}$
$(\mathtt{R}[\mathsf{sID}], \mathtt{SK}[\mathsf{sID}]) := ((\widetilde{\mathsf{ct}}, \mathsf{ct}_i), \mathsf{SK})$
return 1

Oracle $\textsc{Session}_{\mathrm{I}}((i, j) \in [N]^2)$

$\mathsf{cnt} := \mathsf{cnt} + 1, \mathsf{sID} := \mathsf{cnt}$
$(\mathtt{Init}[\mathsf{sID}], \mathtt{Resp}[\mathsf{sID}]) := (i, j)$
$\mathtt{Type}[\mathsf{sID}] := \text{“In”}$
$(\mathsf{ct}_j, \mathsf{K}_j) \leftarrow \mathsf{Encaps}_1(\mathsf{pk}'_j)$
$\mathcal{C}_j := \mathcal{C}_j \cup \{\mathsf{ct}_j\}$
$\widetilde{\mathsf{pk}} := \widetilde{\mathsf{pk}}^*_{\mathsf{sID}}, \ \widetilde{\mathcal{K}} := \widetilde{\mathcal{K}} \cup \{(\widetilde{\mathsf{pk}}, \mathsf{sID})\}$
$\mathtt{Cor}'[\widetilde{\mathsf{pk}}] := 0$
$IV \leftarrow \{0,1\}^\kappa, \varphi \leftarrow \{0,1\}^d$
$\mathsf{st}' := (\widetilde{\mathsf{pk}}, \bot, \mathsf{ct}_j, \mathsf{K}_j)$
$\mathsf{st} := (IV, \varphi), \ \mathtt{St}'_i[IV] := (\varphi, \mathsf{st}')$
$(\mathtt{I}[\mathsf{sID}], \mathtt{St}[\mathsf{sID}]) := ((\widetilde{\mathsf{pk}}, \mathsf{ct}_j), \mathsf{st})$
return $(\mathsf{sID}, (\widetilde{\mathsf{pk}}, \mathsf{ct}_j))$

Oracle $\textsc{Session}_{\mathrm{R}}((i, j) \in [N]^2, (\widetilde{\mathsf{pk}}, \mathsf{ct}_j))$

$\mathsf{cnt} := \mathsf{cnt} + 1, \mathsf{sID} := \mathsf{cnt}$
$(\mathtt{Init}[\mathsf{sID}], \mathtt{Resp}[\mathsf{sID}]) := (i, j)$
$\mathtt{Type}[\mathsf{sID}] := \text{“Re”}$
$\mathsf{K}_j := \mathsf{Decaps}_1(\mathsf{sk}_j, \mathsf{ct}_j)$
if $\mathsf{K}_j = \bot$: **return** $\bot$
if $\exists t$ s.t. $(\widetilde{\mathsf{pk}}, t) \in \widetilde{\mathcal{K}}$
  $\widetilde{\mathsf{ct}} \leftarrow \textsc{Enc}_0(t), \ \widetilde{\mathcal{C}} := \widetilde{\mathcal{C}} \cup \{(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{ct}})\}$
else
  $(\widetilde{\mathsf{ct}}, \widetilde{\mathsf{K}}) \leftarrow \mathsf{Encaps}_0(\widetilde{\mathsf{pk}})$
  $\widetilde{\mathcal{L}} := \widetilde{\mathcal{L}} \cup \{(\widetilde{\mathsf{ct}}, \widetilde{\mathsf{K}})\}$
$(\mathsf{ct}_i, \mathsf{K}_i) \leftarrow \mathsf{Encaps}_1(\mathsf{pk}_i), \mathcal{C}_i := \mathcal{C}_i \cup \{\mathsf{ct}_i\}$
$\mathsf{ctxt} := (\mathsf{pk}'_i, \mathsf{pk}'_j, \widetilde{\mathsf{pk}}, \mathsf{ct}_i, \mathsf{ct}_j, \widetilde{\mathsf{ct}})$
$\mathsf{SK} \leftarrow \mathcal{SK}$
$\mathtt{SK}'[\mathsf{ctxt}] := \mathsf{SK}, \ \mathtt{SK}[\mathsf{sID}] := \mathsf{SK}$
$(\mathtt{I}[\mathsf{sID}], \mathtt{R}[\mathsf{sID}]) := ((\widetilde{\mathsf{pk}}, \mathsf{ct}_j), (\widetilde{\mathsf{ct}}, \mathsf{ct}_i))$
return $(\mathsf{sID}, (\widetilde{\mathsf{ct}}, \mathsf{ct}_i))$

Figure 22: $\mathcal{B}_0$ has access to $\mathrm{O}_{\mathsf{KEM}_0} := \{\textsc{Enc}_0, \textsc{Dec}_0, \textsc{Reveal}_0, \textsc{Corr}_0, \textsc{Check}_0\}$. $\mathcal{A}$ has access to $\mathrm{O} := (\textsc{Session}_{\mathrm{I}}, \textsc{Der}_{\mathrm{I}}, \textsc{Session}_{\mathrm{R}}, \textsc{RevK}, \textsc{Cor}, \textsc{Test})$. Oracles $\textsc{Test}$ and $\textsc{Cor}$ are the same as $\mathbf{G}_{4,b}$ in Figure 18. Oracles G, H, and $\textsc{RevSt}$ are given in Figure 23. Here highlighted lines show how the simulator uses oracles in $\mathrm{O}_{\mathsf{KEM}_0}$ to simulate $\mathbf{G}_{4,b}$ and break $\mathsf{OW\text{-}ChCCA}$ security of KEM_0.

Oracle $\mathsf{G}(\mathsf{k}, IV)$

01 **if** $\exists y$ s.t. $(\mathsf{k}, IV, y) \in \mathcal{L}_{\mathsf{G}}$
02 **return** y
03 $y \leftarrow \{0,1\}^d$
04 **if** $\exists i$ s.t. $\mathsf{k} = \mathsf{k}_i \wedge \mathtt{St}'_i[IV] \neq \bot$
05 $(\varphi, \mathsf{st}') := \mathtt{St}'_i[IV]$
06 $(\widetilde{\mathsf{pk}}, \bot, \mathsf{ct}_j, \mathsf{K}_j) := \mathsf{st}'$
07 $\mathcal{C}_j := \mathcal{C}_j \backslash \{\mathsf{ct}_j\}$
08 Find t s.t. $(\widetilde{\mathsf{pk}}, t) \in \widetilde{\mathcal{K}}$
09 $\widetilde{\mathsf{sk}} := \textsc{Corr}_0(t)$
10 $y := \varphi \oplus (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}, \mathsf{ct}_j, \mathsf{K}_j)$
11 $\mathtt{Cor}'[\widetilde{\mathsf{pk}}] := 1$
12 $\mathcal{L}_{\mathsf{G}} := \mathcal{L}_{\mathsf{G}} \cup \{(\mathsf{k}, IV, y)\}$
13 **return** y

Oracle $\textsc{RevSt}(\mathsf{sID})$

14 **if** $\mathtt{Type}[\mathsf{sID}] \neq$ "In": **return** $\bot$
15 $i := \mathtt{Init}[\mathsf{sID}]$
16 $(IV, \varphi) := \mathtt{St}[\mathsf{sID}]$
17 **if** $\exists y$ s.t. $(\mathsf{k}_i, IV, y) \in \mathcal{L}_{\mathsf{G}}$
18 **if** $\neg\mathtt{Cor}[i] \vee \neg\mathtt{revST}[\mathsf{sID}]$
19 $\mathsf{bad}_{\mathsf{st}} := 1$
20 **output** $(t^*, \widetilde{\mathsf{ct}}^*, \bot)$ and **abort**
21 $\mathtt{revST}[\mathsf{sID}] := 1$
22 **return** $\mathtt{St}[\mathsf{sID}]$

Oracle $\mathsf{H}(\mathsf{ctxt}, \mathsf{K}_1, \mathsf{K}_2, \widetilde{\mathsf{K}})$

23 **if** $\exists \mathsf{SK}$ s.t. $(\mathsf{ctxt}, \mathsf{K}_1, \mathsf{K}_2, \widetilde{\mathsf{K}}, \mathsf{SK}) \in \mathcal{L}_{\mathsf{H}}$ **return** SK
24 **parse** $(\mathsf{pk}_i, \mathsf{pk}_j, \widetilde{\mathsf{pk}}, \mathsf{ct}_1, \mathsf{ct}_2, \widetilde{\mathsf{ct}}) := \mathsf{ctxt}$
25 **if** $\mathtt{SK}'[\mathsf{ctxt}] \neq \bot \wedge \mathsf{Decaps}_1(\mathsf{sk}_i, \mathsf{ct}_1) = \mathsf{K}_1 \wedge \mathsf{Decaps}_1(\mathsf{sk}_j, \mathsf{ct}_2) = \mathsf{K}_2$
26 **if** $(\exists t$ s.t. $(\widetilde{\mathsf{pk}}, t) \in \widetilde{\mathcal{K}} \wedge \textsc{Dec}_0(t, \widetilde{\mathsf{ct}}) = \widetilde{\mathsf{K}}) \vee (\widetilde{\mathsf{ct}}, \widetilde{\mathsf{K}}) \in \widetilde{\mathcal{L}}$
27 $\mathsf{SK} := \mathtt{SK}'[\mathsf{ctxt}]$
28 **if** $\big(\neg\mathtt{Cor}[i] \wedge \mathsf{ct}_1 \in \mathcal{C}_i \wedge \mathsf{K}_1 = \mathsf{Decap}_1(\mathsf{sk}'_i, \mathsf{ct}_1)\big)$
 $\vee \big(\neg\mathtt{Cor}[j] \wedge \mathsf{ct}_2 \in \mathcal{C}_j \wedge \mathsf{K}_2 = \mathsf{Decap}_1(\mathsf{sk}'_j, \mathsf{ct}_2)\big)$
29 $\mathsf{bad}_1 := 1$, **output** $(t^*, \widetilde{\mathsf{ct}}^*, \bot)$ and **abort**
30 **if** $\exists t$ s.t.$(\widetilde{\mathsf{pk}}, t) \in \widetilde{\mathcal{K}} \wedge (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{ct}}) \in \widetilde{\mathcal{C}} \wedge \textsc{Check}_0(t, \widetilde{\mathsf{ct}}, \widetilde{\mathsf{pk}}) = 1 \wedge \neg\mathtt{Cor}'[\widetilde{\mathsf{pk}}]$
31 $\mathsf{bad}_0 := 1$, $t^* := t, \widetilde{\mathsf{ct}}^* := \widetilde{\mathsf{ct}}$, **output** $(t^*, \widetilde{\mathsf{ct}}^*, \widetilde{\mathsf{K}})$ and **abort**
32 $\mathsf{SK} \leftarrow \mathcal{SK}$
33 $\mathcal{L}_{\mathsf{H}} := \mathcal{L}_{\mathsf{H}} \cup \{(\mathsf{ctxt}, \mathsf{K}_1, \mathsf{K}_2, \widetilde{\mathsf{K}}, \mathsf{SK})\}$
34 **return** SK

Figure 23: Oracles G, H, and $\textsc{RevSt}$ in the proof of Lemma E.2. Here highlighted lines show how the simulator uses oracles in $\mathrm{O}_{\mathsf{KEM}_0}$ to simulate $\mathbf{G}_{4,b}$ and break OW-ChCCA security of KEM_0.

KEM key of $\widetilde{\mathsf{ct}}$ as unknown and uses $\textsc{Check}_0$ to search adversary $\mathcal{A}$'s queries to H and recover the key from them to break the OW-ChCCA security of KEM_0.

- In H, $\mathcal{B}_0$ queries $\textsc{Check}_0(t, \widetilde{\mathsf{ct}}, \widetilde{\mathsf{K}})$ to determine if $\mathsf{Decaps}_0(\widetilde{\mathsf{sk}}, \widetilde{\mathsf{ct}}) = \widetilde{\mathsf{K}}$, where $\widetilde{\mathsf{sk}}$ is the private key of $\widetilde{\mathsf{pk}}^*_t$. If $\mathcal{A}$ triggers bad_0 event, then $\mathcal{B}_0$ records and outputs the identity, challenge ciphertext, and corresponding key, and aborts the simulation (see Lines 30 to 31).
- If other bad events (i.e., bad_1 or $\mathsf{bad}_{\mathsf{st}}$) happens, $\mathcal{B}_0$ aborts the simulation with outputting $(t^*, \widetilde{\mathsf{ct}}^*, \bot)$.

Let $(t^*, \widetilde{\mathsf{ct}}^*, \widetilde{\mathsf{K}})$ be $\mathcal{B}_0$'s final output. If $\mathcal{A}$ triggers bad_0 event, then by definition, we have $\mathtt{Cor}'[\widetilde{\mathsf{pk}}^*_{t^*}] = 0$, $(\widetilde{\mathsf{pk}}^*_{t^*}, \widetilde{\mathsf{ct}}^*) \in \widetilde{\mathcal{C}}$, and $\widetilde{\mathsf{K}}$ is the KEM_0 key of $\widetilde{\mathsf{ct}}^*$ wrt $\widetilde{\mathsf{pk}}_{t^*}$. $\mathtt{Cor}'[\widetilde{\mathsf{pk}}^*_{t^*}] = 0$ means that $\mathcal{B}_0$ never queries $\textsc{Corr}_0(t^*)$. $(\widetilde{\mathsf{pk}}^*_{t^*}, \widetilde{\mathsf{ct}}^*) \in \widetilde{\mathcal{C}}$ means that $\widetilde{\mathsf{ct}}^*$ is a challenge ciphertext wrt $\widetilde{\mathsf{pk}}_{t^*}$ (and $\mathcal{B}_0$ never queries $\textsc{Reveal}_0(i^*, \mathsf{ct}^*)$). Therefore, $(t^*, \widetilde{\mathsf{ct}}^*, \widetilde{\mathsf{K}})$ is a valid solution and $\mathcal{B}_0$ wins game $\mathsf{OW\text{-}ChCCA}_{\mathsf{KEM}_0}(\lambda)$ (Figure 3). We also need to count the error term δ_0 into the final bound since KEM_0 is $(1-\delta_0)$-correct. That is, we have

$$\left|\Pr\left[\mathbf{G}^{\mathcal{A}}_{3,b} \Rightarrow 1\right] - \Pr\left[\mathbf{G}^{\mathcal{A}}_{4,b} \Rightarrow 1\right]\right| \leq \Pr\left[\mathsf{bad}_0\right] \leq \mathsf{Adv}^{\mathsf{OW\text{-}ChCCA}}_{\mathcal{B},\mathsf{KEM}_0}(\lambda) + \delta_0,$$

as stated in Lemma E.2.

□

Remark on Lemma E.2. Our protocol requires the OW-ChCCA security of the ephemeral KEM KEM_0, while the ephemeral KEM in Jager et al.'s protocol is non-committing against chosen-plaintext attacks (NC-CPA)

(namely, no query to the decapsulation oracle is allowed). This is because we need a decapsulation oracle to give a tight proof against active adversaries $\mathcal{A}$ who can adaptively change ephemeral ciphertext $\widetilde{\mathsf{ct}}$.

More precisely, imagine type (3) attacks in Table 1, namely, $\mathcal{A}$ corrupts the long-term keys of both initiator i and responder j, and activates a session between i with j. $\mathcal{A}$ forwards the first message from i to j honestly, but for the message from j to i $\mathcal{A}$ adaptively changes $\widetilde{\mathsf{ct}}$. We denote the modified $\widetilde{\mathsf{ct}}$ as $\widetilde{\mathsf{ct}}'$. Now the reduction $\mathcal{B}_0$ has already embedded a challenge public key in $\widetilde{\mathsf{pk}}$ and a challenge ciphertext in (non-modified) $\widetilde{\mathsf{ct}}$, but $\mathcal{B}_0$ must decrypt $\widetilde{\mathsf{ct}}'$, since $\widetilde{\mathsf{ct}}'$ can be an invalid ciphertext and initiator i (i.e., $\textsc{Der}_{\mathrm{I}}$) has to output $\perp$ for invalid ciphertexts. Here $\mathcal{B}_0$ cannot corrupt the secret key of $\widetilde{\mathsf{pk}}$, otherwise the corresponding ciphertext $\widetilde{\mathsf{ct}}$ is not fresh, and according to the definition of $\mathsf{OW\text{-}ChCCA}$ it is not a valid attack. Hence, without the decapsulation oracle $\mathcal{B}_0$ cannot tightly simulate the experiment.

However, for the $\mathsf{NC\text{-}CPA}$ security in [JKRS21], $\mathcal{B}_0$ can corrupt the secret key to simulate the experiment, and the corresponding ciphertexts are still counted as valid. This indicates that the non-committing KEM notion in [JKRS21] (even against chosen-plaintext attacks) seems to be stronger than our $\mathsf{OW\text{-}ChCCA}$ security.

Game $\mathsf{IND\text{-}ECCA}^{\mathcal{A}}_{\mathsf{KEM},\beta}(\lambda)$

$\mathsf{par} \leftarrow \mathsf{Setup}(1^\lambda)$
for $i \in [N] : (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(\mathsf{par})$
$\mathrm{O}_1 := (\textsc{Enc}, \textsc{Dec}, \textsc{Reveal})$
$\mathrm{O}_2 := (\textsc{Corr}, \textsc{Test})$
$\beta' \leftarrow \mathcal{A}^{\mathrm{O}_1,\mathrm{O}_2}(\mathsf{par}, (\mathsf{pk}_i)_{i\in[N]})$
return β'

Oracle $\textsc{Enc}(i)$

$(\mathsf{ct}, K) \leftarrow \mathsf{Encap}(\mathsf{pk}_i)$
$\mathcal{L}_{\textsc{Enc}} := \mathcal{L}_{\textsc{Enc}} \cup \{(i, \mathsf{ct}, K)\}$
return ct

Oracle $\textsc{Dec}(i, \mathsf{ct}')$

if $\exists K'$ s.t. $(i, \mathsf{ct}', K') \in \mathcal{L}_{\textsc{Enc}}$:
  return $\perp$
return $K' \leftarrow \mathsf{Decap}(\mathsf{sk}_i, \mathsf{ct}')$

Oracle $\textsc{Reveal}(i, \mathsf{ct})$

if $(i, \mathsf{ct}) \in \mathcal{L}_{\textsc{Test}}$: **return** $\perp$
if $\exists K : (i, \mathsf{ct}', K) \in \mathcal{L}_{\textsc{Enc}}$:
  $\mathcal{L}_{\textsc{Reveal}} := \mathcal{L}_{\textsc{Reveal}} \cup \{(i, \mathsf{ct})\}$
  return K
return $\perp$

Oracle $\textsc{Corr}(i)$

if $\exists \mathsf{ct}$ s.t. $(i, \mathsf{ct}) \in \mathcal{L}_{\textsc{Test}}$:
  return $\perp$
$\mathcal{L}_{\textsc{Corr}} := \mathcal{L}_{\textsc{Corr}} \cup \{i\}$
return sk_i

Oracle $\textsc{Test}(i, \mathsf{ct})$

if $(i, \mathsf{ct}) \in \mathcal{L}_{\textsc{Test}} \cup \mathcal{L}_{\textsc{Reveal}}$:
  return $\perp$
if $i \in \mathcal{L}_{\textsc{Corr}}$: **return** $\perp$
if $\exists K : (i, \mathsf{ct}, K) \in \mathcal{L}_{\textsc{Enc}}$:
  $\mathcal{L}_{\textsc{Test}} := \mathcal{L}_{\textsc{Test}} \cup \{(i, \mathsf{ct})\}$
  $K_0 := K,\ K_1 \xleftarrow{\$} \mathcal{K}$
  return K_β
return $\perp$

Figure 24: The game IND-ECCA for a key encapsulation mechanism $\mathsf{KEM} := (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encap}, \mathsf{Decap})$.

F From OW-ChCCA to IND-ECCA

We show that OW-ChCCA security implies IND-ECCA security, via a simple transformation. IND-ECCA security has been introduced by Han et al. [HLG21] to show impossibility of tightly secure AKE based on many well-known key encapsulation mechanisms in the standard model. Similar to OW-ChCCA, IND-ECCA is a multi-user multi-challenge notion for key encapsulation mechanisms, where the adversary can access a corruption oracle and make decapsulation queries even for challenge ciphertexts. The important difference is that IND-ECCA security is an indistinguishability-style security notion, and the adversary does not have access to a check oracle $\textsc{Check}$. Our result shows that by hashing the encapsulated key, we can transform any OW-ChCCA secure scheme into an IND-ECCA secure one.

We first define IND-ECCA security formally.

Definition F.1 (IND-ECCA Security). Let $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encap}, \mathsf{Decap})$ be a key encapsulation mechanism and consider the game IND-ECCA defined in Figure 24. We say that KEM is IND-ECCA secure, if for all PPT adversaries $\mathcal{A}$, the following advantage is negligible:

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

In Figure 25 we transform a OW-ChCCA secure $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encap}, \mathsf{Decap})$ into a IND-ECCA secure $\mathsf{KEM}' = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encap}', \mathsf{Decap}')$ using a random oracle $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^\lambda$.

Alg $\mathsf{Encap}'(\mathsf{pk})$

$(\mathsf{ct}, K) \leftarrow \mathsf{Encap}(\mathsf{pk})$
$K' := \mathsf{H}(\mathsf{ct}, K)$
return (ct, K')

Alg $\mathsf{Decap}'(\mathsf{sk}, \mathsf{ct})$

$K \leftarrow \mathsf{Decap}(\mathsf{sk}, \mathsf{ct})$
$K' := \mathsf{H}(\mathsf{ct}, K)$
return K'

Figure 25: The key encapsulation mechanism $\mathsf{KEM}' = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encap}', \mathsf{Decap}')$ for a given key encapsulation mechanism $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encap}, \mathsf{Decap})$. Here, $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^\lambda$ is a random oracle.

Lemma F.2 *Let* $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^\lambda$ *be a random oracle. If* KEM *is* OW-ChCCA *secure, then* KEM′ *is* IND-ECCA *secure.*

Concretely, for any PPT algorithm $\mathcal{A}$ *there is a PPT algorithm* $\mathcal{B}$ *with* $\mathbf{T}(\mathcal{A}) \approx \mathbf{T}(\mathcal{B})$ *and*

$$\mathsf{Adv}^{\mathsf{IND\text{-}ECCA}}_{\mathcal{A},\mathsf{KEM}'}(\lambda) \leq 2 \cdot \mathsf{Adv}^{\mathsf{OW\text{-}ChCCA}}_{\mathcal{B},\mathsf{KEM}}(\lambda).$$

Proof. Let $\mathcal{A}$ be a PPT algorithm. We show the statement using a sequence of games $\mathbf{G}_i$ for $i \in \{0,\dots,3\}$, and use the notation

$$\mathsf{Adv}_i := \Pr[\mathbf{G}_i \Rightarrow 1], \text{ for } i \in \{0,\dots,3\}.$$

The games are presented formally in Figure 26.

Game $\mathbf{G}_0$-$\mathbf{G}_3$

$\mathsf{par} \leftarrow \mathsf{Setup}(1^\lambda)$
for $i \in [N]$: $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(\mathsf{par})$
$\mathrm{O}_1 := (\textsc{Enc}', \textsc{Dec}', \textsc{Reveal}')$
$\mathrm{O}_2 := (\textsc{Corr}', \textsc{Test}')$
$\beta' \leftarrow \mathcal{A}^{\mathrm{O}_1,\mathrm{O}_2}(\mathsf{par}, (\mathsf{pk}_i)_{i\in[N]})$
return β'

Oracle $\textsc{Enc}'(i)$

$(\mathsf{ct}, K) \leftarrow \mathsf{Encap}(\mathsf{pk}_i)$
$K' := \mathsf{H}(\mathsf{ct}, K)$
$\mathcal{L}_K := \mathcal{L}_K \cup \{(i, \mathsf{ct}, K)\}$ // $\mathbf{G}_1$-$\mathbf{G}_2$
$\mathcal{L}_{\textsc{Enc}} := \mathcal{L}_{\textsc{Enc}} \cup \{(i, \mathsf{ct}, K')\}$
return ct

Oracle $\textsc{Dec}'(i, \mathsf{ct}')$

if $\exists K'$ s.t. $(i, \mathsf{ct}', K') \in \mathcal{L}_{\textsc{Enc}}$:
  return $\bot$
return $K' \leftarrow \mathsf{Decap}'(\mathsf{sk}_i, \mathsf{ct}')$

Oracle $\textsc{Reveal}'(i, \mathsf{ct})$

if $(i, \mathsf{ct}) \in \mathcal{L}_{\textsc{Test}}$: **return** $\bot$
if $\exists K$ s.t. $(i, \mathsf{ct}', K) \in \mathcal{L}_{\textsc{Enc}}$:
  $\mathcal{L}_{\textsc{Reveal}} := \mathcal{L}_{\textsc{Reveal}} \cup \{(i, \mathsf{ct})\}$
  $\mathcal{L}_K := \mathcal{L}_K \setminus \{(i', \mathsf{ct}', K) \in \mathcal{L}_K \mid i' = i \wedge \mathsf{ct}' = \mathsf{ct}\}$ // $\mathbf{G}_1$-$\mathbf{G}_2$
  return K
return $\bot$

Oracle $\textsc{Corr}'(i)$

if $\exists \mathsf{ct}$ s.t. $(i, \mathsf{ct}) \in \mathcal{L}_{\textsc{Test}}$: **return** $\bot$
$\mathcal{L}_{\textsc{Corr}} := \mathcal{L}_{\textsc{Corr}} \cup \{i\}$
$\mathcal{L}_K := \mathcal{L}_K \setminus \{(i', \mathsf{ct}, K) \in \mathcal{L}_K \mid i' = i\}$ // $\mathbf{G}_1$-$\mathbf{G}_2$
return sk_i

Oracle $\mathsf{H}(\mathsf{ct}, K)$

if $\exists i$ s.t. $(i, \mathsf{ct}, K) \in \mathcal{L}_K$: // $\mathbf{G}_1$-$\mathbf{G}_2$
  $\mathsf{badQ} := 1$, **abort** // $\mathbf{G}_1$-$\mathbf{G}_2$
if $h[\mathsf{ct}, K] = \bot$:
  $h[\mathsf{ct}, K] \xleftarrow{\$} \{0,1\}^\lambda$
return $h[\mathsf{ct}, K]$

Oracle $\textsc{Test}'(i, \mathsf{ct})$

if $(i, \mathsf{ct}) \in \mathcal{L}_{\textsc{Test}} \cup \mathcal{L}_{\textsc{Reveal}}$:
  return $\bot$
if $i \in \mathcal{L}_{\textsc{Corr}}$: **return** $\bot$
if $\exists K'$ s.t. $(i, \mathsf{ct}, K') \in \mathcal{L}_{\textsc{Enc}}$:
  $\mathcal{L}_{\textsc{Test}} := \mathcal{L}_{\textsc{Test}} \cup \{(i, \mathsf{ct})\}$
  $K'_0 := K'$, $K'_1 \xleftarrow{\$} \mathcal{K}$
  return K'_0 // $\mathbf{G}_0$-$\mathbf{G}_1$
  return K'_1 // $\mathbf{G}_2$-$\mathbf{G}_3$
return $\bot$

Figure 26: The games $\mathbf{G}_0$-$\mathbf{G}_3$ in the proof of Lemma F.2. Lines with highlighted comments are only executed in the corresponding games.

Game $\mathbf{G}_0$: The game $\mathbf{G}_0$ is defined to be $\mathbf{G}_0 := \mathsf{IND\text{-}ECCA}^{\mathcal{A}}_{\mathsf{KEM}',0}(\lambda)$. That is, $\mathbf{G}_0$ is the real IND-ECCA game with bit $\beta = 0$. Recall that in this game, parameters par and keys $(\mathsf{pk}_i, \mathsf{sk}_i)$ are generated. Then, $\mathcal{A}$ is run on input $\mathsf{par}, (\mathsf{pk}_i)_{i\in[N]}$ with access to oracles $\textsc{Enc}', \textsc{Dec}', \textsc{Reveal}', \textsc{Corr}', \textsc{Test}'$. Here, oracle $\textsc{Enc}'$ runs algorithm Encap'. This means that it first runs $(\mathsf{ct}, K) \leftarrow \mathsf{Encap}(\mathsf{pk})$, and then sets $K' := \mathsf{H}(\mathsf{ct}, K)$. It outputs ct to $\mathcal{A}$. As $\beta = 0$, oracle $\textsc{Test}'$ returns the key K' that is computed in the corresponding call to oracle $\textsc{Enc}'$. Additionally, $\mathcal{A}$ gets access to random oracle H. Then, algorithm $\mathcal{A}$ outputs a bit β'. The game outputs β'. By definition, we have

$$\mathsf{Adv}_0 := \Pr\left[\mathsf{IND\text{-}ECCA}^{\mathcal{A}}_{\mathsf{KEM}',0}(\lambda) \Rightarrow 1\right].$$

Game $\mathbf{G}_1$: In game $\mathbf{G}_1$, we add a bad event badQ, and let the game abort if this event occurs. Namely, consider a query of the form $\textsc{Enc}'(i)$, and let (ct, K) be the output of algorithm Encap as described above. We say that badQ occurs, if $\mathcal{A}$ queries $\mathsf{H}(\mathsf{ct}, K)$ before any query of the form $\textsc{Corr}'(i)$ or $\textsc{Reveal}'(i, \mathsf{ct})$ for these i, ct, that does not return $\perp$. In the code, we formally model this via a list $\mathcal{L}_K$, where we insert tuples (i, ct, K) during queries of the form $\textsc{Enc}'(i)$, and remove them in corresponding queries (that do not output $\perp$) of the form $\textsc{Corr}'(i)$ or $\textsc{Reveal}'(i, \mathsf{ct})$. It is clear that the distinguishing advantage between $\mathbf{G}_0$ and $\mathbf{G}_1$ can be bounded by the probability of event badQ.

Reduction $\mathcal{B}$ gets as input par and keys $(\mathsf{pk}_i)_i$. It also gets access to oracles $\textsc{Enc}, \textsc{Dec}, \textsc{Reveal}, \textsc{Corr}, \textsc{Check}$. Then, $\mathcal{B}$ runs $\mathcal{A}$ on input par and keys $(\mathsf{pk}_i)_i$ with access to oracles $\textsc{Enc}', \textsc{Dec}', \textsc{Reveal}', \textsc{Corr}', \textsc{Test}'$, and a random oracle H. We describe how $\mathcal{B}$ simulates these oracles:

- $\textsc{Corr}'(i)$: $\mathcal{B}$ calls $\textsc{Corr}(i)$ and returns the result.
- $\textsc{Reveal}'(i, \mathsf{ct})$: $\mathcal{B}$ calls $K \leftarrow \textsc{Reveal}(i, \mathsf{ct})$. If $K = \perp$, it returns $\perp$. Otherwise, it returns $\mathsf{H}(\mathsf{ct}, K)$.
- $\textsc{Dec}'(i, \mathsf{ct})$: $\mathcal{B}$ calls $K \leftarrow \textsc{Dec}(i, \mathsf{ct})$. If $K = \perp$, it returns $\perp$. Otherwise, it returns $\mathsf{H}(\mathsf{ct}, K)$.
- $\textsc{Enc}'(i)$: $\mathcal{B}$ calls $\textsc{Enc}(i)$ and gets a ciphertext ct. It returns ct. If there is a previous query ct', K to H with $\mathsf{ct} = \mathsf{ct}'$ and $\textsc{Check}(i, \mathsf{ct}, K) = 1$, $\mathcal{B}$ sets $\mathsf{badQ} := 1$, outputs (i, ct, K) to its experiment and terminates.
- $\textsc{Test}(i, \mathsf{ct})$: $\mathcal{B}$ outputs $\perp$ if $\mathbf{G}_1$ would do so (note that it can keep track of the necessary lists), and otherwise it outputs $K' \xleftarrow{\$} \{0,1\}^\lambda$.
- $\mathsf{H}(\mathsf{ct}, K)$: If ct is a previous output of a query $\textsc{Enc}'(i)$, and the oracles $\textsc{Reveal}'(i, \mathsf{ct})$ and $\textsc{Corr}'(i)$ were never queried without returning $\perp$, and $\textsc{Check}(i, \mathsf{ct}, K) = 1$, $\mathcal{B}$ sets $\mathsf{badQ} := 1$, outputs (i, ct, K) to its experiment and terminates.

If $\mathcal{B}$ did not yet terminate when $\mathcal{A}$ outputs bit b, $\mathcal{B}$ outputs $\perp$ and terminates.

Note that the view of $\mathcal{A}$ is identical to its view in $\mathbf{G}_1$ until $\mathcal{B}$ terminates. Especially, $\mathcal{B}$ terminates with an output other than $\perp$ exactly if badQ occurs, and as long as badQ did not yet occur the values returned by oracle $\textsc{Test}$ are uniformly random in $\mathbf{G}_1$. If $\mathcal{B}$ terminates with an output other than $\perp$, the $\mathsf{OW\text{-}ChCCA}$ game outputs 1. Therefore, we have

$$|\mathsf{Adv}_1 - \mathsf{Adv}_2| \leq \mathsf{Adv}^{\mathsf{OW\text{-}ChCCA}}_{\mathcal{B},\mathsf{KEM}}(\lambda).$$

Game $\mathbf{G}_2$: In game $\mathbf{G}_2$, we switch bit β to $\beta = 1$. That means that in queries to oracle $\textsc{Test}'$, we return a random key $K'_1 \xleftarrow{\$} \mathcal{K}$ instead of the key $K'_0 := K'$, where $K' = \mathsf{H}(\mathsf{ct}, K)$ was computed in the corresponding call of the form $\textsc{Enc}'(i)$. We claim that the view of $\mathcal{A}$ does not change from $\mathbf{G}_1$ to $\mathbf{G}_2$. To see this, recall that $\textsc{Test}'(i, \mathsf{ct})$ outputs $\perp$, if $\mathcal{A}$ made a query $\textsc{Corr}'(i)$ or $\textsc{Reveal}'(i, \mathsf{ct})$ before. Also, whenever the oracles $\textsc{Corr}'(i)$ or $\textsc{Reveal}'(i, \mathsf{ct})$ are called afterwards, they output $\perp$. By the change introduced in the previous game, this means that for the queries $\textsc{Test}'(i, \mathsf{ct})$ that do not output $\perp$, we can assume that $\mathcal{A}$ never queries $\mathsf{H}(\mathsf{ct}, K)$ during the entire game. Therefore, keys K'_0 and K'_1 are distributed identically. We have

$$\mathsf{Adv}_1 = \mathsf{Adv}_2.$$

Game $\mathbf{G}_3$: Game $\mathbf{G}_3$ is the same as game $\mathbf{G}_2$, but we do not longer abort on event badQ. That is, we revert the change that we introduced from $\mathbf{G}_0$ to $\mathbf{G}_1$. A similar argument shows that

$$|\mathsf{Adv}_2 - \mathsf{Adv}_3| \leq \mathsf{Adv}^{\mathsf{OW\text{-}ChCCA}}_{\mathcal{B},\mathsf{KEM}}(\lambda).$$

Now, notice that $\mathbf{G}_3$ is equivalent to game $\mathsf{IND\text{-}ECCA}^{\mathcal{A}}_{\mathsf{KEM}',1}(\lambda)$. Therefore, we have

$$\mathsf{Adv}_3 = \Pr\left[\mathsf{IND\text{-}ECCA}^{\mathcal{A}}_{\mathsf{KEM}',1}(\lambda) \Rightarrow 1\right].$$

The statement follows. □

G Omitted Formal Details of Section 5.2

Definition G.1 (Public-Key Encryption Scheme). A public-key encryption scheme is a tuple of PPT algorithms $\mathsf{PKE} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{par}$ takes as input the security parameter 1^λ, and outputs global system parameters par. We assume that par implicitly define a ciphertext space $\mathcal{C} = \mathcal{C}_{\mathsf{par}}$, and a message space $\mathcal{M} = \mathcal{M}_{\mathsf{par}}$.
- $\mathsf{Gen}(\mathsf{par}) \to (\mathsf{pk}, \mathsf{sk})$ takes as input parameters par, and outputs a public key pk and a secret key sk.
- $\mathsf{Enc}(\mathsf{pk}, \mathsf{m}) \to \mathsf{ct}$ takes as input a public key pk and a message $\mathsf{m} \in \mathcal{M}$, and outputs a ciphertext $\mathsf{ct} \in \mathcal{C}$.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to \mathsf{m}$ is deterministic, takes as input a secret key sk and a ciphertext $\mathsf{ct} \in \mathcal{C}$, and outputs a message $\mathsf{m} \in \mathcal{M} \cup \{\perp\}$.

We say that PKE is ρ-correct, if for every $\mathsf{par} \in \mathsf{Setup}(1^\lambda)$, and every message m, the following probability is at least ρ:

$$\Pr\left[\mathsf{m} = \mathsf{m}' \mid (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{par}), \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{m}), \mathsf{m}' \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct})\right].$$

Definition G.2 (SIM-BiSO-CCA Security). Let $\mathsf{PKE} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a public-key encryption scheme and consider the games REAL-BiSO-CCA and IDEAL-BiSO-CCA defined in Figure 27. We say that PKE is SIM-BiSO-CCA secure, if for all PPT algorithms $\mathcal{A}$, there exists a PPT algorithm $\mathcal{S}$, such that for every PPT algorithm $\mathcal{D}$ the following advantage is negligible:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{SIM\text{-}BiSO\text{-}CCA}}_{\mathcal{A},\mathcal{S},\mathcal{D},\mathsf{PKE}}(\lambda) := |&\Pr\left[\mathsf{REAL\text{-}BiSO\text{-}CCA}^{\mathcal{A},\mathcal{D}}_{\mathsf{PKE}}(\lambda) \Rightarrow 1\right] \\ &- \Pr\left[\mathsf{IDEAL\text{-}BiSO\text{-}CCA}^{\mathcal{D}}_{\mathsf{KEM},\mathcal{S}}(\lambda) \Rightarrow 1\right]|.\end{aligned}$$

Game $\mathsf{REAL\text{-}BiSO\text{-}CCA}^{\mathcal{A},\mathcal{D}}_{\mathsf{PKE}}(\lambda)$

$\mathsf{par} \leftarrow \mathsf{Setup}(1^\lambda)$
for $i \in [N] : (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(\mathsf{par})$
$(\mathcal{M}, St) \leftarrow \mathcal{A}^{\mathrm{DEC}}\left(\mathsf{par}, (\mathsf{pk}_i)_{i\in[N]}\right)$
$(M_1, \ldots, M_N) := M \leftarrow \mathcal{M}$
for $(i, j) \in [N] \times [\mu] :$
  $\rho_{i,j} \leftarrow \mathcal{R},\ \mathsf{m}_{i,j} := M_i[j]$
  $\mathsf{ct}_{i,j} := \mathsf{Enc}(\mathsf{pk}_i, \mathsf{m}_{i,j}; \rho_{i,j})$
$C := (\mathsf{ct}_{1,1}, \ldots, \mathsf{ct}_{N,\mu})$
$o \leftarrow \mathcal{A}^{\mathrm{DEC},\mathrm{CORR},\mathrm{OPEN}}(St, C)$
return $\mathcal{D}(M, \mathcal{L}_{\mathrm{CORR}}, \mathcal{L}_{\mathrm{OPEN}}, o)$

Oracle $\mathrm{CORR}(i)$
$\mathcal{L}_{\mathrm{CORR}} := \mathcal{L}_{\mathrm{CORR}} \cup \{i\}$
return sk_i

Oracle $\mathrm{OPEN}(i, j)$
$\mathcal{L}_{\mathrm{OPEN}} := \mathcal{L}_{\mathrm{OPEN}} \cup \{(i, j)\}$
return $(\mathsf{m}_{i,j}, \rho_{i,j})$

Game $\mathsf{IDEAL\text{-}BiSO\text{-}CCA}^{\mathcal{D}}_{\mathsf{KEM},\mathcal{S}}(\lambda)$

$\mathsf{par} \leftarrow \mathsf{Setup}(1^\lambda)$
$(\mathcal{M}, St) \leftarrow \mathcal{S}(\mathsf{par})$
$(M_1, \ldots, M_N) := M \leftarrow \mathcal{M}$
for $(i, j) \in [N] \times [\mu] :$
  $\mathsf{len}_{i,j} := |M_i[j]|,\ \mathsf{m}_{i,j} := M_i[j]$
$L := (\mathsf{len}_{1,1}, \ldots, \mathsf{len}_{N,\mu})$
$o \leftarrow \mathcal{S}^{\mathrm{CORR}',\mathrm{OPEN}'}(St, L)$
return $\mathcal{D}(M, \mathcal{L}_{\mathrm{CORR}}, \mathcal{L}_{\mathrm{OPEN}}, o)$

Oracle $\mathrm{DEC}(i, \mathsf{ct})$
if $\exists j$ s.t. $\mathsf{ct}_{i,j} = \mathsf{ct} :$ **return** $\perp$
return $\mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct})$

Oracle $\mathrm{CORR}'(i)$
$\mathcal{L}_{\mathrm{CORR}} := \mathcal{L}_{\mathrm{CORR}} \cup \{i\}$
return $(\mathsf{m}_{i,j})_j$

Oracle $\mathrm{OPEN}'(i, j)$
$\mathcal{L}_{\mathrm{OPEN}} := \mathcal{L}_{\mathrm{OPEN}} \cup \{(i, j)\}$
return $\mathsf{m}_{i,j}$

Figure 27: The games REAL-BiSO-CCA and IDEAL-BiSO-CCA for a given public-key encryption scheme $\mathsf{PKE} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with randomness space $\mathcal{R}$.

Lemma G.3 *Let* $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^\lambda \times \{0,1\}^\lambda$ *and* $\mathsf{G}\colon \{0,1\}^* \to \{0,1\}^\lambda$ *be random oracles. If* KEM *is* OW-ChCCA *secure, has deterministic ciphertext derivation, and has* $h = \omega(\log \lambda)$ *bits of ciphertext entropy, then* PKE *is* SIM-BiSO-CCA *secure.*

Game $\mathbf{G}_0$-$\mathbf{G}_4$

$\mathsf{par} \leftarrow \mathsf{Setup}(1^\lambda)$
for $i \in [N] : (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(\mathsf{par})$
$(\mathcal{M}, St) \leftarrow \mathcal{A}^{\textsc{Dec},\mathsf{H},\mathsf{G}}\left(\mathsf{par}, (\mathsf{pk}_i)_{i\in[N]}\right),\ (M_1, \ldots, M_N) := M \leftarrow \mathcal{M}$
for $(i,j) \in [N] \times [\mu] :$
  $K_{i,j} \xleftarrow{\$} \mathcal{K},\ \mathsf{m}_{i,j} := M_i[j],\ \mathsf{ct}_{i,j,0} := \widehat{\mathsf{Encap}}(\mathsf{pk}_i, K_{i,j})$
  if $\exists K$ s.t. $h[\mathsf{ct}_{i,j,0}, K] \neq \bot :$ **abort** // $\mathbf{G}_1$-$\mathbf{G}_4$
  if $\exists (i,j) \neq (i',j') \in [N] \times [\mu]$ s.t. $\mathsf{ct}_{i',j',0} = \mathsf{ct}_{i,j,0} :$ **abort** // $\mathbf{G}_1$-$\mathbf{G}_4$
  $(k^{(e)}_{i,j}, k^{(m)}_{i,j}) := \mathsf{H}(\mathsf{ct}_{i,j,0}, K_{i,j}),\quad \mathsf{ct}_{i,j,1} := k^{(e)}_{i,j} \oplus \mathsf{m}_{i,j}$ // $\mathbf{G}_0$-$\mathbf{G}_1$
  $(\mathsf{ct}_{i,j,1}, k^{(m)}_{i,j}) \xleftarrow{\$} \{0,1\}^\lambda \times \{0,1\}^\lambda$ // $\mathbf{G}_2$-$\mathbf{G}_4$
  $\mathsf{ct}_{i,j,2} := \mathsf{G}(k^{(m)}_{i,j}, \mathsf{ct}_{i,j,1}),\quad \mathsf{ct}_{i,j} := (\mathsf{ct}_{i,j,0}, \mathsf{ct}_{i,j,1}, \mathsf{ct}_{i,j,2})$
$C := (\mathsf{ct}_{1,1}, \ldots, \mathsf{ct}_{N,\mu}),\quad o \leftarrow \mathcal{A}^{\textsc{Dec},\textsc{Corr},\textsc{Open},\mathsf{H},\mathsf{G}}(St, C)$
return $\mathcal{D}(M, \mathcal{L}_{\textsc{Corr}}, \mathcal{L}_{\textsc{Open}}, o)$

Oracle $\mathsf{H}(\mathsf{ct}, K)$

if $h[\mathsf{ct}, K] = \bot :$
  $h[\mathsf{ct}, K] \xleftarrow{\$} \{0,1\}^\lambda \times \{0,1\}^\lambda$
  if $\exists (i,j) \in [N] \times [\mu]$ s.t. $(\mathsf{ct}, K) = (\mathsf{ct}_{i,j,0}, K_{i,j}) \wedge i \notin \mathcal{L}_{\textsc{Corr}} \wedge (i,j) \notin \mathcal{L}_{\textsc{Open}} :$
    $\mathsf{badOW} := 1$, **abort** // $\mathbf{G}_4$
  if $\exists (i,j) \in [N] \times [\mu]$ s.t. $(\mathsf{ct}, K) = (\mathsf{ct}_{i,j,0}, K_{i,j}) :$ // $\mathbf{G}_2$-$\mathbf{G}_4$
    $h[\mathsf{ct}, K] := (\mathsf{ct}_{i,j,1} \oplus \mathsf{m}_{i,j}, k^{(m)}_{i,j})$ // $\mathbf{G}_2$-$\mathbf{G}_4$
return $h[\mathsf{ct}, K]$

Oracle $\textsc{Dec}(i, \mathsf{ct})$

if $\exists j \in [\mu]$ s.t. $\mathsf{ct}_{i,j} = \mathsf{ct} :$ **return** $\bot$
let $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1, \mathsf{ct}_2)$
if $\exists j \in [\mu]$ s.t. $\mathsf{ct}_0 = \mathsf{ct}_{i,j,0}$ // $\mathbf{G}_3$-$\mathbf{G}_4$
  $\wedge\ h[\mathsf{ct}_{i,j,0}, K_{i,j}] = \bot$ // $\mathbf{G}_3$-$\mathbf{G}_4$
  $\wedge\ \mathsf{G}(k^{(m)}_{i,j}, \mathsf{ct}_1) = \mathsf{ct}_2 :$ // $\mathbf{G}_3$-$\mathbf{G}_4$
  $\mathsf{badMAC} := 1$, **abort** // $\mathbf{G}_3$-$\mathbf{G}_4$
$K := \mathsf{Decap}(\mathsf{sk}_i, \mathsf{ct}_0)$
$(k^{(e)}, k^{(m)}) := \mathsf{H}(\mathsf{ct}_0, K)$
if $\mathsf{G}(k^{(m)}, \mathsf{ct}_1) \neq \mathsf{ct}_2 :$ **return** $\bot$
return $\mathsf{m} := \mathsf{ct}_1 \oplus k^{(e)}$

Oracle $\textsc{Open}(i, j)$

$\mathcal{L}_{\textsc{Open}} := \mathcal{L}_{\textsc{Open}} \cup \{(i,j)\}$
return $(\mathsf{m}_{i,j}, K_{i,j})$

Oracle $\textsc{Corr}(i)$

$\mathcal{L}_{\textsc{Corr}} := \mathcal{L}_{\textsc{Corr}} \cup \{i\}$
return sk_i

Oracle $\mathsf{G}(w, \mathsf{ct})$

if $g[w, \mathsf{ct}] = \bot :$
  $g[w, \mathsf{ct}] \xleftarrow{\$} \{0,1\}^\lambda$
return $g[w, \mathsf{ct}]$

Figure 28: The games $\mathbf{G}_0$-$\mathbf{G}_4$ used in the proof of Lemma G.3. Lines with highlighted comments are only executed in the corresponding games.

Concretely, for any PPT algorithm $\mathcal{A}$, there is a PPT algorithms $\mathcal{S}$, such that $\mathbf{T}(\mathcal{S}) \approx \mathbf{T}(\mathcal{A})$, and for every PPT algorithm $\mathcal{D}$ there is a PPT algorithm $\mathcal{B}$ with $\mathbf{T}(\mathcal{B}) \approx \mathbf{T}(\mathcal{A}) + \mathbf{T}(\mathcal{D})$ and

$$\mathsf{Adv}^{\mathsf{SIM\text{-}BiSO\text{-}CCA}}_{\mathcal{A},\mathcal{S},\mathcal{D},\mathsf{PKE}}(\lambda) \leq \frac{Q_{\mathsf{H}} N \mu}{2^h} + \frac{Q_{\textsc{Dec}} Q_{\mathsf{G}}}{2^h} + \mathsf{Adv}^{\mathsf{OW\text{-}ChCCA}}_{\mathcal{B},\mathsf{KEM}}(\lambda),$$

where $Q_{\textsc{Dec}}, Q_{\mathsf{H}}, Q_{\mathsf{G}}$ denotes the number of queries of $\mathcal{A}$ to oracles $\textsc{Dec}, \mathsf{H}, \mathsf{G}$, respectively.

Proof. The statement is proven via a sequence of games as formally given in Figure 28.

We start with game $\mathbf{G}_0 := \mathsf{REAL\text{-}BiSO\text{-}CCA}^{\mathcal{A},\mathcal{D}}_{\mathsf{PKE}}(\lambda)$, and make changes towards a game that can be simulated. For ease of notation, we define

$$\mathsf{Adv}_i := \Pr\left[\mathbf{G}_i \Rightarrow 1\right],\ \text{for } i \in \{0, \ldots, 4\}.$$

Game $\mathbf{G}_0$: Game $\mathbf{G}_0$ is defined as the real attack game $\mathsf{REAL\text{-}BiSO\text{-}CCA}$ We recall this game to fix some notation. First, parameters par and keys $(\mathsf{pk}_i, \mathsf{sk}_i)$ for $i \in [N]$ are generated. Then, $\mathcal{A}$ is run on input par and $(\mathsf{pk}_i)_{i\in[N]}$ with access to a decryption oracle $\textsc{Dec}$ and random oracles H and G. Random oracles H and G are simulated using maps $h[\cdot]$ and $g[\cdot]$ in the standard lazy manner. Adversary $\mathcal{A}$ outputs a distribution $\mathcal{M}$, from which messages $\mathsf{m}_{i,j}$ for all $(i,j) \in [N] \times [\mu]$ are sampled. For each such $(i,j) \in [N] \times [\mu]$,

Alg $\mathcal{B}^{\text{ENC}',\text{DEC}',\text{REVEAL}',\text{CORR}',\text{CHECK}'}(\mathsf{par}, (\mathsf{pk}_i)_{i\in[N]})$

$(\mathcal{M}, St) \leftarrow \mathcal{A}^{\text{DEC},\mathsf{H},\mathsf{G}}(\mathsf{par}, (\mathsf{pk}_i)_{i\in[N]})$, $(M_1, \ldots, M_N) := M \leftarrow \mathcal{M}$
for $(i, j) \in [N] \times [\mu]$:
  $\mathsf{m}_{i,j} := M_i[j]$, $\mathsf{ct}_{i,j,0} \leftarrow \text{ENC}'(i)$
  if $\exists K$ s.t. $h[\mathsf{ct}_{i,j,0}, K] \neq \bot$: **abort**
  if $\exists (i, j) \neq (i', j') \in [N] \times [\mu]$ s.t. $\mathsf{ct}_{i',j',0} = \mathsf{ct}_{i,j,0}$: **abort**
  $(\mathsf{ct}_{i,j,1}, k^{(m)}_{i,j}) \xleftarrow{\$} \{0,1\}^\lambda \times \{0,1\}^\lambda$
  $\mathsf{ct}_{i,j,2} := \mathsf{G}(k^{(m)}_{i,j}, \mathsf{ct}_{i,j,1})$, $\mathsf{ct}_{i,j} := (\mathsf{ct}_{i,j,0}, \mathsf{ct}_{i,j,1}, \mathsf{ct}_{i,j,2})$
$C := (\mathsf{ct}_{1,1}, \ldots, \mathsf{ct}_{N,\mu})$, $o \leftarrow \mathcal{A}^{\text{DEC},\text{CORR},\text{OPEN},\mathsf{H},\mathsf{G}}(St, C)$
return $\mathcal{D}(M, \mathcal{L}_{\text{CORR}}, \mathcal{L}_{\text{OPEN}}, o)$

Oracle $\mathsf{H}(\mathsf{ct}, K)$

if $h[\mathsf{ct}, K] = \bot$:
  $h[\mathsf{ct}, K] \xleftarrow{\$} \{0,1\}^\lambda \times \{0,1\}^\lambda$
  if $\text{CHECK}'(i, \mathsf{ct}, K) = 1$:
    if $\exists (i, j) \in [N] \times [\mu]$ s.t. $\mathsf{ct} = \mathsf{ct}_{i,j,0} \wedge i \notin \mathcal{L}_{\text{CORR}} \wedge (i, j) \notin \mathcal{L}_{\text{OPEN}}$:
      $\mathsf{badOW} := 1$, **return** (i, ct, K) **to** $\mathsf{OW\text{-}ChCCA}_{\mathsf{KEM}}$
    if $\exists (i, j) \in [N] \times [\mu]$ s.t. $(\mathsf{ct}, K) = (\mathsf{ct}_{i,j,0}, K_{i,j})$:
    $h[\mathsf{ct}, K] := (\mathsf{ct}_{i,j,1} \oplus \mathsf{m}_{i,j}, k^{(m)}_{i,j})$
return $h[\mathsf{ct}, K]$

Oracle $\text{DEC}(i, \mathsf{ct})$

if $\exists j \in [\mu]$ s.t. $\mathsf{ct}_{i,j} = \mathsf{ct}$: **return** $\bot$
let $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1, \mathsf{ct}_2)$
if $\exists j \in [\mu]$ s.t. $\mathsf{ct}_0 = \mathsf{ct}_{i,j,0} \wedge \mathsf{G}(k^{(m)}_{i,j}, \mathsf{ct}_1) = \mathsf{ct}_2$: $\mathsf{badMAC} := 1$, **abort**
if $\exists j \in [\mu]$ s.t. $\mathsf{ct}_0 = \mathsf{ct}_{i,j,0}$: **return** $\bot$
$K \leftarrow \text{DEC}'(i, \mathsf{ct}_0)$, $(k^{(e)}, k^{(m)}) := \mathsf{H}(\mathsf{ct}_0, K)$
if $\mathsf{G}(k^{(m)}, \mathsf{ct}_1) \neq \mathsf{ct}_2$: **return** $\bot$
return $\mathsf{m} := \mathsf{ct}_1 \oplus k^{(e)}$

Oracle $\text{OPEN}(i, j)$

$\mathcal{L}_{\text{OPEN}} := \mathcal{L}_{\text{OPEN}} \cup \{(i, j)\}$
$K_{i,j} \leftarrow \text{REVEAL}'(i, \mathsf{ct}_{i,j,0})$
return $(\mathsf{m}_{i,j}, K_{i,j})$

Oracle $\text{CORR}(i)$

$\mathcal{L}_{\text{CORR}} := \mathcal{L}_{\text{CORR}} \cup \{i\}$
$\mathsf{sk}_i \leftarrow \text{CORR}'(i)$
for $j \in [\mu]$:
  $K_{i,j} := \mathsf{Decap}(\mathsf{sk}_i, \mathsf{ct}_{i,j,0})$
return sk_i

Figure 29: The reduction $\mathcal{B}$ in the proof of Lemma G.3, running in the game $\mathsf{OW\text{-}ChCCA}_{\mathsf{KEM}}$. Oracle G is provided as in game $\mathbf{G}_3$.

the game samples $K_{i,j} \xleftarrow{\$} \mathcal{K}$ and computes a ciphertext $\mathsf{ct}_{i,j}$. In the concrete scheme at hand, $\mathsf{ct}_{i,j}$ has the form $\mathsf{ct}_{i,j} := (\mathsf{ct}_{i,j,0}, \mathsf{ct}_{i,j,1}, \mathsf{ct}_{i,j,2})$, where $\mathsf{ct}_{i,j,0} := \widehat{\mathsf{Encap}}(\mathsf{pk}_i, K_{i,j})$, $(k^{(e)}_{i,j}, k^{(m)}_{i,j}) := \mathsf{H}(\mathsf{ct}_{i,j,0}, K_{i,j})$, $\mathsf{ct}_{i,j,1} := k^{(e)}_{i,j} \oplus \mathsf{m}_{i,j}$, and $\mathsf{ct}_{i,j,2} := \mathsf{G}(k^{(m)}_{i,j}, \mathsf{ct}_{i,j,1})$. Adversary $\mathcal{A}$ gets all ciphertexts $\mathsf{ct}_{i,j}$ and access to the previous oracles, as well as a corruption oracle CORR, and an opening oracle OPEN. The final output o of $\mathcal{A}$, the messages M, and its queries $\mathcal{L}_{\text{CORR}}, \mathcal{L}_{\text{OPEN}}$ to oracles CORR, OPEN are given to $\mathcal{D}$. Then, the game outputs whatever $\mathcal{D}$ outputs. Note that we use the deterministic ciphertext derivation property of KEM here. We have

$$\mathsf{Adv}_0 = \Pr\left[\mathsf{REAL\text{-}BiSO\text{-}CCA}^{\mathcal{A},\mathcal{D}}_{\mathsf{PKE}}(\lambda) \Rightarrow 1\right].$$

Game $\mathbf{G}_1$: Game $\mathbf{G}_1$ is as $\mathbf{G}_0$, but we introduce a bad event and let the game abort if this bad event occurs. Namely, the game aborts if for some $(i, j) \in [N] \times [\mu]$, after running $\mathsf{ct}_{i,j,0} := \widehat{\mathsf{Encap}}(\mathsf{pk}_i, K_{i,j})$, there is a K such that value $h[\mathsf{ct}_{i,j,0}, K]$ is already defined, i.e. $\mathcal{A}$ queried $\mathsf{H}(\mathsf{ct}_{i,j,0}, K)$ before, or there is some other tuple $(i', j') \in [N] \times [\mu]$ such that $\mathsf{ct}_{i',j',0} = \mathsf{ct}_{i,j,0}$. As the former condition is implied by the latter, it is sufficient to bound the probability of the former. Clearly, we can bound the probability of this event using the ciphertext entropy of KEM and a union bound over all such $(i, j) \in [N] \times [\mu]$ and all

random oracle queries. We get

$$|\mathsf{Adv}_0 - \mathsf{Adv}_1| \leq \frac{Q_{\mathsf{H}} N \mu}{2^h}.$$

Game $\mathbf{G}_2$: Game $\mathbf{G}_2$ is as $\mathbf{G}_1$, but we change the way we compute the ciphertexts $\mathsf{ct}_{i,j}$ for all $(i,j) \in [N] \times [\mu]$ and simulate random oracle H. Recall that before, we computed $(k^{(e)}_{i,j}, k^{(m)}_{i,j}) := \mathsf{H}(\mathsf{ct}_{i,j,0}, K_{i,j})$ and $\mathsf{ct}_{i,j,1} := k^{(e)}_{i,j} \oplus \mathsf{m}_{i,j}$. From now on, we sample $(\mathsf{ct}_{i,j,1}, k^{(m)}_{i,j}) \xleftarrow{\$} \{0,1\}^\lambda \times \{0,1\}^\lambda$ and continue as before. Then, if $\mathcal{A}$ queries $\mathsf{H}(\mathsf{ct}_{i,j,0}, K_{i,j})$ afterwards, the game programs the random oracle as $h[\mathsf{ct}_{i,j,0}, K_{i,j}] := (\mathsf{ct}_{i,j,1} \oplus \mathsf{m}_{i,j}, k^{(m)}_{i,j})$. Due to the previous change, this does not change the view of the adversary. We have

$$\mathsf{Adv}_1 = \mathsf{Adv}_2.$$

Game $\mathbf{G}_3$: In $\mathbf{G}_3$, we introduce another bad event and let the game abort if it occurs. Namely, we define the bad event badMAC as follows. Consider a decryption query $\textsc{Dec}(i, \mathsf{ct})$ for $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1, \mathsf{ct}_2)$, which does not return $\perp$ due to $\mathsf{ct} = \mathsf{ct}_{i,j}$ for some $j \in [\mu]$. Event badMAC occurs in such a query, if we have $\mathsf{ct}_0 = \mathsf{ct}_{i,j,0}$ for some $j \in [\mu]$ with $h[\mathsf{ct}_{i,j,0}, K_{i,j}] = \perp$ and $\mathsf{G}(k^{(m)}_{i,j}, \mathsf{ct}_1) = \mathsf{ct}_2$.

Now, we have to bound the probability of badMAC. To this end, consider a fixed decryption query, for which $\mathsf{ct}_0 = \mathsf{ct}_{i,j,0}$ for some $(i,j) \in [N] \times [\mu]$ with $h[\mathsf{ct}_{i,j,0}, K_{i,j}] = \perp$. First, consider the case $\mathsf{ct}_1 = \mathsf{ct}_{i,j,1}$. Then, we know that

$$\mathsf{ct}_{i,j,2} = \mathsf{G}(k^{(m)}_{i,j}, \mathsf{ct}_{i,j,1}) = \mathsf{G}(k^{(m)}_{i,j}, \mathsf{ct}_1) = \mathsf{ct}_2.$$

Thus, we have $\mathsf{ct} = \mathsf{ct}_{i,j}$, and the oracle would have returned $\perp$, a contradiction. For the second case, assume $\mathsf{ct}_1 \neq \mathsf{ct}_{i,j,1}$. We know that the only information that $\mathcal{A}$ obtained about $k^{(m)}_{i,j}$ is $\mathsf{G}(k^{(m)}_{i,j}, \mathsf{ct}_{i,j,1})$. This is because we assume $h[\mathsf{ct}_{i,j,0}, K_{i,j}] = \perp$. As $\mathsf{ct}_1 \neq \mathsf{ct}_{i,j,1}$, the hash value $\mathsf{G}(k^{(m)}_{i,j}, \mathsf{ct}_{i,j,1})$ does not reveal information about $\mathsf{G}(k^{(m)}_{i,j}, \mathsf{ct}_1)$, and so $\mathsf{G}(k^{(m)}_{i,j}, \mathsf{ct}_1) = \mathsf{ct}_2$ can only occur with probability at most $Q_{\mathsf{G}}/2^\lambda$. A union bound over all decryption queries leads to

$$|\mathsf{Adv}_2 - \mathsf{Adv}_3| \leq \frac{Q_{\textsc{Dec}} Q_{\mathsf{G}}}{2^h}.$$

Game $\mathbf{G}_4$: In $\mathbf{G}_4$, we introduce another bad event and let the game abort if it occurs. Namely, the bad event badOW occurs, if for some $(i,j) \in [N] \times [\mu]$, adversary $\mathcal{A}$ queries $\mathsf{H}(\mathsf{ct}_{i,j,0}, K_{i,j})$ after getting all the ciphertexts, and before ever calling $\textsc{Corr}(i)$ or $\textsc{Open}(i,j)$. Clearly, we have

$$|\mathsf{Adv}_3 - \mathsf{Adv}_4| \leq \Pr[\mathsf{badOW}].$$

We bound the probability of event badOW using a reduction $\mathcal{B}$ that breaks the $\mathsf{OW\text{-}ChCCA}$ security of KEM, assuming badOW occurs. We formally present $\mathcal{B}$ in Figure 29. It gets as input parameters par and keys $(\mathsf{pk}_i)_{i \in [N]}$. It also gets access to oracles $\textsc{Enc}', \textsc{Dec}', \textsc{Reveal}', \textsc{Corr}', \textsc{Check}'$ of game $\mathsf{OW\text{-}ChCCA}_{\mathsf{KEM}}$. Then, it runs $\mathcal{A}$ on input par and $(\mathsf{pk}_i)_{i \in [N]}$, as in $\mathbf{G}_3$. It provides random oracles H and G as in $\mathbf{G}_3$, and a decryption oracle $\textsc{Dec}$. Here, queries of the form $\textsc{Dec}(i, \mathsf{ct})$ with $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1, \mathsf{ct}_2)$ are simulated by $\mathcal{B}$ by first forwarding (i, ct_0) to oracle $\textsc{Dec}$ and getting key K in return. Then, $\mathcal{B}$ continues answering this query as in $\mathbf{G}_3$. Once $\mathcal{A}$ outputs $\mathcal{M}$, $\mathcal{B}$ samples the $\mathsf{m}_{i,j}$ as in $\mathbf{G}_3$. It computes each $\mathsf{ct}_{i,j,0}$ by calling $\textsc{Enc}'(i)$. It aborts if there is some K such that $h[\mathsf{ct}_{i,j,0}, K] \neq \perp$ (cf. $\mathbf{G}_1$). Then, it computes the remaining parts of $\mathsf{ct}_{i,j}$ as in $\mathbf{G}_3$. Note that due to the change in $\mathbf{G}_2$, the value $K_{i,j}$ is not needed for that. Next, $\mathcal{B}$ passes all ciphertexts $\mathsf{ct}_{i,j}$ to $\mathcal{A}$, and provides access to oracles $\mathsf{H}, \mathsf{G}, \textsc{Dec}, \textsc{Corr}, \textsc{Open}$. Random oracle G is simulated honestly as in $\mathbf{G}_3$, and $\textsc{Corr}$ is simulated by forwarding between $\mathcal{A}$ and $\textsc{Corr}'$. The other oracles are simulated as follows:

- $\textsc{Corr}(i)$: $\mathcal{B}$ calls $\textsc{Corr}'(i)$, which returns sk_i. Then, $\mathcal{B}$ computes $K_{i,j} := \mathsf{Decap}(\mathsf{sk}_i, \mathsf{ct}_{i,j,0})$ for all $j \in [\mu]$, and returns sk_i to $\mathcal{A}$.
- $\textsc{Open}(i,j)$: $\mathcal{B}$ calls $\textsc{Reveal}'(i, \mathsf{ct}_{i,j,0})$, which returns $K_{i,j}$. Then, $\mathcal{B}$ returns $(\mathsf{m}_{i,j}, K_{i,j})$ to $\mathcal{A}$.
- $\mathsf{H}(\mathsf{ct}, K)$: If the hash value has to be defined, $\mathcal{B}$ first checks if badOW occurs. To do that, it first checks if $\mathsf{ct} = \mathsf{ct}_{i,j,0}$ for some i,j for which $\mathcal{A}$ never queried $\textsc{Corr}(i)$ or $\textsc{Open}(i,j)$. If this is the case, it runs oracle $\textsc{Check}'(i, \mathsf{ct}, K)$. If the oracle returns 1, $\mathcal{B}$ terminates by outputting (i, ct, K). Otherwise, it simulates the random oracle as in $\mathbf{G}_3$.

- $\textsc{Dec}(i, \mathsf{ct})$ for $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1, \mathsf{ct}_2)$: As in $\mathbf{G}_3$, $\mathcal{B}$ returns $\perp$ if $\mathsf{ct} = \mathsf{ct}_{i,j}$ for some $j \in [\mu]$. Otherwise, $\mathcal{B}$ first checks if $\mathsf{ct}_0 = \mathsf{ct}_{i,j,0}$ for some $j \in [\mu]$. If it is, it checks if $\mathsf{G}(k_{i,j}^{(m)}, \mathsf{ct}_1) = \mathsf{ct}_2$. If this holds, $\mathcal{B}$ aborts. It is easy to see that, because $\mathcal{B}$ did not yet terminate, this happens exactly when event badMAC occurs. If $\mathcal{B}$ did not abort, and $\mathsf{ct}_0 = \mathsf{ct}_{i,j,0}$ for some $j \in [\mu]$, $\mathcal{B}$ returns $\perp$. Note that in this case, oracle $\textsc{Dec}$ would also return $\perp$ in $\mathbf{G}_3$. Otherwise, $\mathcal{B}$ computes K by calling $\textsc{Dec}'(i, \mathsf{ct}_0)$. This will never return $\perp$, as $\mathsf{ct}_0 \neq \mathsf{ct}_{i,j,0}$ for all $j \in [\mu]$. Then, it uses K to complete the simulation of the query as in $\mathbf{G}_3$.

If $\mathcal{B}$ did not yet terminate when $\mathcal{A}$ returns its final output o, $\mathcal{B}$ outputs $\perp$. First, we see that $\mathcal{B}$ perfectly simulates $\mathbf{G}_3$ for $\mathcal{A}$ until it terminates Further, if $\mathcal{B}$ terminates, then it breaks the $\mathsf{OW\text{-}ChCCA}$ security of KEM. Further, $\mathcal{B}$ terminates whenever event badOW occurs. This shows that

$$|\mathsf{Adv}_3 - \mathsf{Adv}_4| \leq \mathsf{Adv}^{\mathsf{OW\text{-}ChCCA}}_{\mathcal{B},\mathsf{KEM}}(\lambda).$$

Finally, it is easy to see that $\mathbf{G}_4$ can be simulated by a simulator $\mathcal{S}$. This is because the messages $\mathsf{m}_{i,j}$ are only needed after oracle queries of the form $\textsc{Corr}(i)$ or $\textsc{Open}(i, j)$. Thus, $\mathcal{S}$ can internally provide $\mathbf{G}_4$ for $\mathcal{A}$ and forward its output. We have

$$\mathsf{Adv}_3 = \Pr\left[\mathsf{IDEAL\text{-}BiSO\text{-}CCA}^{\mathcal{D}}_{\mathsf{KEM},\mathcal{S}}(\lambda) \Rightarrow 1\right],$$

finishing the proof.

□