Key Exchange with Tight (Full) Forward Secrecy via Key Confirmation

Jiaxin Pan[1], Doreen Riepel[2], and Runzhi Zeng[3]

[1] University of Kassel, Kassel, Germany
[2] University of California San Diego, La Jolla, USA
[3] Norwegian University of Science and Technology, Trondheim, Norway
jiaxin.pan@uni-kassel.de, driepel@ucsd.edu, runzhi.zeng@ntnu.no

Abstract. Weak forward secrecy (wFS) of authenticated key exchange (AKE) protocols is a passive variant of (full) forward secrecy (FS). A natural mechanism to upgrade from wFS to FS is the use of key confirmation messages which compute a message authentication code (MAC) over the transcript. Unfortunately, Gellert, Gjøsteen, Jacobson and Jager (GGJJ, CRYPTO 2023) show that this mechanism inherently incurs a loss proportional to the number of users, leading to an overall non-tight reduction, even if wFS was established using a tight reduction.

Inspired by GGJJ, we propose a new notion, called one-way verifiable weak forward secrecy (OW-VwFS), and prove that OW-VwFS can be transformed *tightly* to FS using key confirmation in the random oracle model (ROM). To implement our generic transformation, we show that several tightly wFS AKE protocols additionally satisfy our OW-VwFS notion tightly. We highlight that using the recent lattice-based protocol from Pan, Wagner, and Zeng (CRYPTO 2023) can give us the first lattice-based tightly FS AKE via key confirmation in the classical random oracle model. Besides this, we also obtain a Decisional-Diffie-Hellman-based protocol that is considerably more efficient than the previous ones.

Finally, we lift our study on FS via key confirmation to the quantum random oracle model (QROM). While our security reduction is overall non-tight, it matches the best existing bound for wFS in the QROM (Pan, Wagner, and Zeng, ASIACRYPT 2023), namely, it is square-root- and session-tight. Our analysis is in the multi-challenge setting, and it is more realistic than the single-challenge setting as in Pan et al..

Keywords: Authenticated key exchange · forward secrecy · key confirmation · tight security · (quantum) random oracles.

1 Introduction

Forward secrecy (FS) is an essential security requirement for authenticated key exchange (AKE) protocols. It states that even if an active adversary corrupts a user's long-term secret key, all session keys agreed before should remain secret to the adversary. A weaker form of FS is called weak FS (wFS), where an adversary is not allowed to perform active attacks, namely, it does not actively interfere with the protocol transcripts of the session that it attacks.

Key confirmation is simple and arguably the most efficient way in achieving FS and has been used in many works, e.g., [Kra05,FGSW16,CCG+19]. Essentially, it generically transforms an AKE protocol with wFS to FS. More precisely, two parties firstly run a wFS AKE protocol to agree on a session key k, and then they exchange key confirmation messages derived from k. These messages are usually message authentication codes (MAC) on the protocol transcripts using k as the MAC key. Apart from key confirmation, one can use a digital signature scheme to sign a passively secure key exchange protocol as in the signed Diffie-Hellman protocol [PQR22,GJ18] to provide FS. Considering that using a MAC or hash function is much more efficient than digital signatures, the signature-based approach is often inefficient and less desirable.

Security Models for AKE. Defining the security for AKE protocols is a complex task, and there are many different security models for AKE (e.g., [BR94,CK01,LLM07]). In this paper, we consider active adversaries that can modify, drop, or inject some messages. Moreover, they may adaptively corrupt users' long-term secret keys via Corr oracle and reveal session keys via Reveal oracle. Some of the

models even allow adversaries to learn ephemeral states (which are usually randomness in generating protocol messages) via Rev-State oracle. We formalize key secrecy via Test, where an adversary $\mathcal{A}$ chooses a *fresh* session, receives either a real or random key for it, and shall distinguish between the two. We consider the single-bit guessing, multi-challenge security, namely, $\mathcal{A}$ can query Test multiple times and each time Test responds using the same bit in deciding real or random. Composability for this notion was initially proven for password-based key exchange [AFP05], and we refer to [JKRS21] for further discussion on why this is the realistic and meaningful notion. For forward secrecy, keys of these Test-sessions must be computed before Corr is queried to either parties of a Test-session. Depending on the type of forward secrecy, freshness is defined differently. If it is wFS, then $\mathcal{A}$ must perform only passive attacks on this fresh session. Otherwise, $\mathcal{A}$ can perform active attacks, for instance, modify or inject some messages.

Security Loss for FS via Key Confirmation. The complexity of AKE models makes it challenging to prove security of an AKE protocol, in particular, giving tight security proofs for AKE. The security of modern cryptographic protocols is often proven by reductions. A reduction $\mathcal{R}$ uses an adversary $\mathcal{A}$ against protocol Π to break the security of the underlying primitive P. By doing so, we can conclude the concrete security bound, $\varepsilon_{\mathcal{A}} \leq \ell \cdot \varepsilon_{\mathcal{R}}$, where $\varepsilon_{\mathcal{A}}$ and $\varepsilon_{\mathcal{R}}$ are the success probability of $\mathcal{A}$ and $\mathcal{R}$, respectively. ℓ is called the security loss. Assuming $\mathcal{A}$ and $\mathcal{R}$ have roughly the same running time, if ℓ is a small constant, we say protocol Π has tight security, and non-tight security, otherwise. A tight security reduction is highly desirable, since it allows protocols to be instantiated with optimal parameters without compensation for the security loss.

A natural question to ask is whether the key confirmation approach preserves the tightness of the underlying wFS AKE. Due to its high efficiency, it would be ideal to have an affirmative answer to this question, since it means that we do not need to increase the security parameter of the wFS AKE to compensate any security loss.

Intuitively, there should not be a tightness loss when going from wFS to FS, which was even falsely claimed by the work of Cohn-Gordon et al. [CCG+19] previously. At CRYPTO 2023, Gellert, Gjøsteen, Jacobsen, and Jager (GGJJ) [GGJJ23] identified a flaw in [CCG+19] and proposed a fix by using a selective variant of wFS (called selective key secrecy in [GGJJ23]). The selective wFS is essentially the same as wFS, except that an adversary $\mathcal{A}$ has to select a user of which $\mathcal{A}$ will not corrupt the long-term secret key. Unfortunately, when we construct a reduction $\mathcal{R}$ to prove FS based on this selective wFS, $\mathcal{R}$ has to guess the non-corrupted user, which leads to a security loss of $O(\mu)$ where μ is the maximal number of users. This security loss is proven to be inherent (and thus optimal) in [GGJJ23] when starting from a wFS AKE with key indistinguishability.

However, a linear loss in the number of users is undesirable, since in the real world the number of users can be massive. According to the impossibility result in [CCG+19], it seems inherent to have this security loss. Hence, it motivates us to propose a different modularization that potentially requires strong security for the underlying wFS AKE in achieving tight FS.

1.1 Our Contribution I: Tight Forward Secrecy via Key Confirmation

We revise the security proof for the wFS-to-FS transformation.

Tight FS from Verifiable wFS. We propose a new variant of wFS, called One-Wayness against key Verification attacks and weak Forward Secrecy (OW-VwFS). In the OW-VwFS security game, an adversary has the same capability as in the usual wFS game, but additionally it can verify whether a session key is the valid one of a particular session. Hence, the adversary capability of OW-VwFS is stronger than that of wFS and it is the main reason why we bypass the optimality result from Gellert et al. [GGJJ23]. In terms of security goals, OW-VwFS is weaker than wFS, namely, OW-VwFS only requires an adversary cannot compute the session key of a fresh session, while wFS requires a session key to be indistinguishable from a random key.

Using key confirmation, we prove that OW-VwFS *tightly* implies FS in the random oracle model. Our transformation is the same as the standard wFS-to-FS transformation, but ours preserves the tightness of the underlying OW-VwFS protocol, and it enables tight FS in contrast to the selective notion in [GGJJ23]. An important consequence of our work is that the future AKE design can aim at OW-VwFS, since its transformation to FS is the same as the standard wFS-to-FS one, but tightness-preserving. Moreover,

our analysis considers security against (ephemeral) state reveals. Such a strong form of attacks was not considered in the work of Gellert et al. [GGJJ23], which is why we bypass their impossibility.

Constructing (Tightly) Verifiable wFS. Furthermore, we show that several tightly wFS protocols satisfy our new OW-VwFS notion tightly, in particular, the lattice-based protocol of Pan, Wagner, and Zeng [PWZ23a][4]. Subsequently, this yields the *first* AKE protocol with tight FS from lattices.

Essentially, we show that a One-Way Checkable against Chosen-Ciphertext Attacks (OW-ChCCA) [PWZ23a] secure key encapsulation mechanism (KEM) tightly implies a OW-VwFS AKE protocol. Once again, our analysis allows adversaries to reveal ephemeral state in the AKE protocol. Roughly speaking, the OW-ChCCA game is a multi-user, multi-challenge variant of the standard IND-CCA game: Besides the oracles provided by the standard IND-CCA security, it allows adversaries to corrupt some of the user's decryption keys and decrypt some of the challenge ciphertexts, and, most importantly, it allows an adversary to check if a key is valid with respect to a ciphertext. The adversary goal is to invert a fresh challenge ciphertext. As shown in [PWZ23a], we can construct OW-ChCCA KEM tightly from the Decisional Diffie-Hellman (DDH) and Learning-With-Errors (LWE) assumptions, respectively. As a technical note, our proof requires only a slightly weaker version of OW-ChCCA, where adversaries are not allowed to ask for a decryption, but to verify whether a ciphertext can be decapsulated.

Efficiency Comparison among DDH-based Protocols. Besides having the first lattice-based AKE with tight FS, we also obtain the most efficient DDH-based protocol against state reveal attacks. In Table 1, we compare efficiency among well-known DDH-based AKE with tight or "optimal" tight FS (namely, with security loss $O(\mu)$) to show the practicality of our work. Our estimation focuses on communication and computation complexity for both parties to agree on a session key. For computation complexity, we only count the number of exponentiations, since they are the most costly operations. For concrete efficiency, we instantiate the protocols at 128-bit security and assume that the number of users $\mu \approx 2^{30}$. This is about the number of monthly active users in a social media app[5]. We instantiate the fully tight protocols with a NIST P256 curve and "optimal" tight ones with a NIST P384 (since they require a 158-bit hard DLog assumption). Our benchmarks for an exponentiation in a P256 and P384 are 0.5 ms and 1 ms, using Apple M1 Max, 32GB of RAM and macOS Ventura 13.3.1 (a).

We observe that the DDH-based non-committing KEM in [JKRS21] is tightly OW-ChCCA secure (cf. [PWZ23a, Footnote 1]). Our analysis shows that the wFS JKRS in [JKRS21] is tightly OW-VwFS, and after adding key confirmation to the wFS JKRS in [JKRS21] we get $\mathsf{JKRS}_{\mathsf{KC}}$ that is tightly FS. According to Table 1, our tight security proofs allow one to implement $\mathsf{JKRS}_{\mathsf{KC}}$ with about 30% shorter transcripts and 50% faster speed than the one with the "optimal", non-tight proofs in [GGJJ23] at 128-bit security. Considering security against State Reveals, $\mathsf{JKRS}_{\mathsf{KC}}$ is the most efficient DDH-based protocol, due to our tight security proofs. It is worth mentioning that the $\mathsf{CCGJJ}_{\mathsf{KC}}$ has shorter protocol transcripts, but it is insecure under State Reveals.

Interestingly, although the signature-based JKRS uses relatively inefficient primitives as signatures, its tight security proof allows an instantiation that is slightly more efficient than the non-tight, signature-less JKRS (namely, $\mathsf{JKRS}_{\mathsf{KC}}$ with proofs in [GGJJ23]).

Relation to the work of Gellert et al. [GGJJ23]. We circumvent the impossibility result of Gellert et al. [GGJJ23] by using a different wFS notion, OW-VwFS, and random oracles. As discussed earlier, the key checking oracle makes our notion stronger. The security definition in [GGJJ23] does not have such an oracle and thus their impossibility result does not apply to our proof. At the same time, we opted for the weakest definition which allows a tight reduction (i.e., one-wayness and also no Reveal oracle), which makes our definition and that of [GGJJ23] incomparable (neither implies the other). Moreover, their impossibility is in the standard model, while ours is in the random oracle model. For these reasons, our results do not contradict the impossibility result in [GGJJ23], but rather provides an alternative way to prove security while enabling full tightness.

One might wonder whether the impossibility result persists at all when moving to the (programmable) random oracle model. The authors of [GGJJ23] provide some indication that this is the case and we want

[4] Their lattice-based protocol is almost tight (similar to [CW13]), since it needs to lose a factor of $O(\lambda)$ to the LWE assumption, where λ is the security parameter. We call it tight as well, but specify the concrete loss in our theorems and proofs.

[5] Cf. https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/.

Protocol	Comm. ($\mathbb{G}, H$, Sign, other)	Bytes	Exp.	Time (ms)	#Msg.	State Reveal	Security loss
TLS 1.3 [DG21,DJ21]	$(2, 2, 2, 512)$	384	32	16	3	no	$O(1)$
GJ [GJ18]	$(2, 1, 2, 0)$	288	32	16	3	no	$O(1)$
LLGW [LLGW20]	$(3, 0, 2, 0)$	288	35	17.5	2	no	$O(1)$
JKRS [JKRS21]	$(5, 1, 1, 0)$	288	29	14.5	2	yes	$O(1)$
PQR [PQR22]	$(2, 0, 2, 0)$	256	32	16	2	no	$O(1)$
$\mathsf{CCGJJ}_{\mathsf{KC}}$ [CCG+19]	(2,2,0,0)	160	8	8	3	no	$O(\mu)$
$\mathsf{JKRS}_{\mathsf{KC}}$ [GGJJ23]	(5,2,0,0)	304	15	15	3	yes	$O(\mu)$
$\mathsf{JKRS}_{\mathsf{KC}}$ (Ours)	(5,2,0,0)	224	15	7.5	3	yes	$O(1)$

Table 1. Comparison of Diffie-Hellman-based AKE protocols with (tight or "optimal" tight) FS. Concrete efficiency is estimated for 128-bit security. "$\mathsf{JKRS}_{\mathsf{KC}}$ [GGJJ23]" is transforming the implicitly authenticated JKRS [JKRS21] via key confirmation. We estimate its efficiency, according to the "optimal", non-tight security bound by Gellert et al. [GGJJ23]. The last row is the same construction as the second last one, but with our tight security proof (cf. Theorem 4). In the upper arrows, schemes are using signatures, and we estimate the concrete bytes with the most efficient signature scheme in [DGJL21]. **Comm.** counts values exchanged during the protocol execution. $\mathbb{G}$ counts the number of group elements, H the number of hashes or MACs, 'Sign' the number of signatures, and 'other' the additional data in bits. **Bytes** counts total data in bytes by instantiating $\mathbb{G}$ with NIST P256 or P384 (for the non-tight $\mathsf{JKRS}_{\mathsf{KC}}$). **Exp.** counts the total numbers of exponentiation (which is the most costly computation in an AKE protocol) from both parties in agreeing a session key, and **Time** is the estimated time of computing those exponentiation in milliseconds.

to give a short discussion as well. For this note that a reduction (in the key indistinguishability game) will be able to simulate key confirmation messages by picking tags at random. If the session key is known at a later point in time (e.g., because the adversary asks to reveal the session key which allows the reduction to reveal the key as well), the reduction can patch the random oracle accordingly. However, if the adversary does not explicitly ask to reveal the key and instead manages to query the random oracle on an unknown session key, the reduction will not able to identify such a query to make the simulation consistent. This explains why we require one-way security (such a strategy requires the adversary to compute the session key) and the key verification oracle to identify such a query without tightness loss.

1.2 Our Contribution II: Forward Secrecy via Key Confirmation in the QROM

Our second contribution is proposing the first security proof for FS via key confirmation in the quantum random oracle model (QROM) [BDF+11], where a quantum adversary can have quantum access to the hash function. Our analysis considers the KEM-based AKE protocol (via key confirmation) and assumes a Multi-User, Multi-Challenge Chosen-Ciphertext Attacks (MUC-CCA) KEM and a Multi-Challenge CCA (MC-CCA) KEM. The main reason of doing so is that we do not know how to tightly prove OW-VwFS implies FS in the QROM, since it will trigger the Oneway-to-Hiding Lemma [Unr14] and lead to a square-root-loss such as $\sqrt{\varepsilon}$, where ε is the advantage of breaking the underlying KEM. We still think that our tight lattice-based protocol in the classical ROM is interesting, since it is the first protocol with tight FS from post-quantum assumptions. Of course, one may alternatively rephrase our analysis in the classical ROM with the suitable KEMs, but it may lower the readability. More importantly, our OW-VwFS notion is more generic and gives more freedom to designers to construct their OW-VwFS protocols that will lead to FS in a tightness-preserving manner.

Our security bound in the QROM is unfortunately non-tight. More precisely, ignoring the statistically negligible terms, **our security bound for FS** in the QROM is

$$\varepsilon_{\mathrm{FS}}^{\mathsf{our}} \leq O(\mu) \cdot \varepsilon_{\text{MC-CCA}} + O(1) \cdot \varepsilon_{\text{MUC-CCA}}. \tag{1}$$

where μ is the number of users, $\varepsilon_{\text{MC-CCA}}$ is the advantage of MC-CCA, and $\varepsilon_{\text{MUC-CCA}}$ is that of MUC-CCA. It matches the best known bound for wFS in the QROM proposed by Pan, Wagner, and Zeng (PWZ) [PWZ23b]. In this sense our FS bound preserves the tightness of the KEM-based AKE protocol with wFS. It is worth mentioning that, as shown in [PWZ23b], we can tightly instantiate MC-CCA and MUC-CCA from the LWE assumption in the QROM.

We also improve PWZ's analysis in the sense that their analysis considers only one single Test query, but our security bound (as stated in Equation (1)) is established in the context of multiple

challenges, where an adversary is allowed to query TEST multiple times. The multi-challenge setting is more realistic and well-established for public-key primitives [BBM00,FHKP13,GHKW16,CCG+19], since in the real world, an adversary usually wants to attack multiple instances of a primitives. Although the security bound of PWZ can be extended to the multi-TEST setting with a multiplicative factor t (which is the number of TEST-queries), ours does not need to lose such a factor. In practice, t can be up to the total number of established sessions, which can be much larger than the number of users. We stress that the analysis of PWZ is only for wFS, and transforming it to FS, it may lose another multiplicative factor μ by applying the analysis of GGJJ [GGJJ23]. Hence, combining the analysis of PWZ and GGJJ leads to a bound for FS in the QROM as

$$\varepsilon_{\mathrm{FS}}^{\mathsf{PWZ\ \&\ GGJJ}} \leq O(\mu^2 t) \cdot \varepsilon_{\text{MC-CCA}} + O(\mu t) \cdot \varepsilon_{\text{MUC-CCA}}. \tag{2}$$

Strictly speaking, the bound above is only an estimation and not theoretically sound, since the analysis of GGJJ is in the classical setting. Their bound may change, if an adversary can query the hash function or key derivation function with a quantum state.

MORE RELATED WORK IN THE QROM. Another work on the KEM-based AKE protocol in the QROM is due to Hövelmanns, Kiltz, Schäge, and Unruh [HKSU20], and it has a square-root-loss, namely, its security bound is

$$O(S^2 + S \cdot \mu) \cdot \left(\varepsilon_{\text{CPA}} + \sqrt{Q \cdot \varepsilon_{\text{CPA}}}\right), \tag{3}$$

where S, μ, and Q are the numbers of total sessions, users, and random oracle queries, respectively, and ε_{CPA} is the advantage of breaking the underlying CPA secure PKE. Similar to the work of PWZ, Equation (3) is only for wFS and in the single-TEST setting. Upgrading to FS in the multi-TEST setting requires an additional multiplicative loss in μt. It is usually less desirable to have the square-root-loss as in Equation (3), since it reduces the security guarantee of the underlying PKE in half.

2 Preliminaries

For $n \in \mathbb{N}$, let $[n] = \{1, \ldots, n\}$. For a finite set $\mathcal{S}$, we denote the sampling of a uniform random element x by $x \xleftarrow{\$} \mathcal{S}$. By $[\![B]\!]$ we denote the bit that is 1 if the evaluation of the Boolean statement B is **true** and 0 otherwise.

ALGORITHMS. For an algorithm $\mathcal{A}$ which takes x as input, we denote its computation by $y \coloneqq \mathcal{A}(x)$ if $\mathcal{A}$ is deterministic, and $y \leftarrow \mathcal{A}(x)$ if $\mathcal{A}$ is probabilistic. We assume all the algorithms (including adversaries) in this paper to be probabilistic unless stated differently. We denote an algorithm $\mathcal{A}$ with access to an oracle O by $\mathcal{A}^{\mathrm{O}}$. In terms of running time, if a reduction's running time t' is dominated by that of an adversary t (more precisely, $t' = t + s$ where $s \ll t$), we write $t' \approx t$.

GAMES. We use code-based games [BR06] to present our definitions and proofs. We implicitly assume all Boolean flags to be initialized to 0 (**false**), numerical variables to 0, sets to $\varnothing$ and strings to $\perp$. We make the convention that a procedure terminates once it has returned an output. $G^{\mathcal{A}} \Rightarrow b$ denotes the final (Boolean) output b of game G running adversary $\mathcal{A}$, and if $b = 1$ we say $\mathcal{A}$ wins G. The randomness in $\Pr[G^{\mathcal{A}} \Rightarrow 1]$ is over all random coins in game G. More generically, we write $\Pr[\mathsf{Event} : G]$ to denote the probability that Event happens in the game G. If the context is clear, we simply write it as $\Pr[\mathsf{Event}]$. Within a procedure, "**abort**" means that we terminate the run of an adversary $\mathcal{A}$.

3 Three-Message Authenticated Key Exchange

We recall the AKE security model from [JKRS21] and adapt it to three-message protocols. A three-message key exchange protocol $\mathsf{AKE} \coloneqq (\mathsf{Setup}, \mathsf{Gen}_{\mathsf{AKE}}, \mathsf{Init}_{\mathsf{I}}, \mathsf{Init}_{\mathsf{R}}, \mathsf{Der}_{\mathsf{I}}, \mathsf{Der}_{\mathsf{R}})$ consists of five algorithms which are executed interactively by two parties as shown in Figure 1.

Setup is the setup algorithm for the system parameters. We denote the party which initiates the session by P_i and the party which responds to the session by P_r. The key generation algorithm $\mathsf{Gen}_{\mathsf{AKE}}$ outputs a key pair $(\mathsf{pk}, \mathsf{sk})$ for one party. The initiator's initialization algorithm $\mathsf{Init}_{\mathsf{I}}$ inputs the initiator's long-term secret key sk_i and the responder's long-term public key pk_r, and outputs a message $m_{i,1}$ and the initiator's state st_i. The responder's initialization algorithm $\mathsf{Init}_{\mathsf{R}}$ inputs the responder's long-term

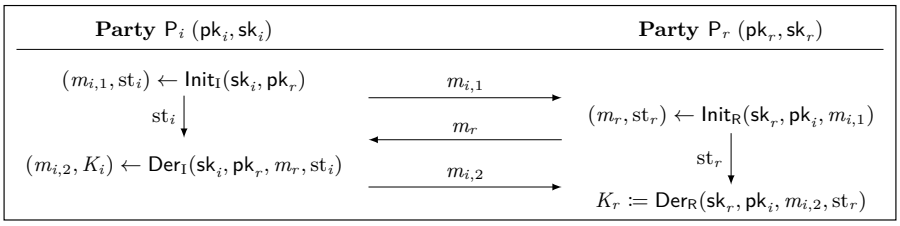

Fig. 1. Running a three-message AKE protocol between two parties.

secret key sk_r and the initiator's long-term public key pk_i, and outputs a message m_r and the responder's state st_r. The initiator's derivation algorithm Der_I takes as input sk_i, pk_r, a message m_r and the state st_i. It computes the final message $m_{i,2}$ and a session key K. The responder's derivation algorithm Der_R takes as input the sk_r, pk_i, a message $m_{i,2}$ and the state st_r. It computes the session key K. Here K can be $\perp$ meaning that the session is rejected during the execution. Correctness of an AKE protocol states that an honest execution between two parties should yield the same session key.

Definition 1 (Correctness of three-message AKE). *Let* $\mathsf{AKE} \coloneqq (\mathsf{Setup}, \mathsf{Gen}_\mathsf{AKE}, \mathsf{Init}_\mathrm{I}, \mathsf{Init}_\mathsf{R}, \mathsf{Der}_\mathrm{I}, \mathsf{Der}_\mathsf{R})$ *be a three-message AKE protocol. We say* AKE *is* ρ*-correct if*

$$\Pr\left[K_i = K_r \neq \perp : \begin{array}{l} \mathsf{par} \leftarrow \mathsf{Setup}(1^\lambda), \\ (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}_\mathsf{AKE}(\mathsf{par}), (\mathsf{pk}_r, \mathsf{sk}_r) \leftarrow \mathsf{Gen}_\mathsf{AKE}(\mathsf{par}), \\ (m_{i,1}, \mathrm{st}_i) \leftarrow \mathsf{Init}_\mathrm{I}(\mathsf{sk}_i, \mathsf{pk}_r), \\ (m_r, \mathrm{st}_r) \leftarrow \mathsf{Init}_\mathsf{R}(\mathsf{sk}_r, \mathsf{pk}_i, m_{i,1}), \\ (m_{i,2}, K_i) \leftarrow \mathsf{Der}_\mathrm{I}(\mathsf{sk}_i, \mathsf{pk}_r, m_r, \mathrm{st}_i), \\ K_r \coloneqq \mathsf{Der}_\mathsf{R}(\mathsf{sk}_r, \mathsf{pk}_i, m_{i,2}, \mathrm{st}_r) \end{array}\right] \geq \rho \ ,$$

where the probability is taken over the randomness of $\mathsf{Setup}, \mathsf{Gen}_\mathsf{AKE}, \mathsf{Init}_\mathrm{I}, \mathsf{Init}_\mathsf{R}$*, and* Der_I*.*

We give a security game written in pseudocode focusing on (full) forward secrecy, rather than implicit or explicit authentication. We refer readers to [DFW20] for more details on different types of authentication for key exchange protocols, and their connections to forward secrecy in [GGJJ23].

Execution Environment. We consider μ parties $\mathsf{P}_1, \ldots, \mathsf{P}_\mu$ with long-term key pairs $(\mathsf{pk}_n, \mathsf{sk}_n)$, $n \in [\mu]$. When two parties A and B want to communicate, the initiator, say, A first creates a session. To identify this session, we increase the global identification number sID and assign the current state of sID to identify this session owned by A. The state of sID will increase after every assignment. Moreover, a message will be sent to the responder. The responder then similarly creates a corresponding session which is assigned the current state of sID. Hence each conversation includes two sessions. We then define variables in relation to the identifier sID:

- `Init`[sID] $\in [\mu]$ denotes the initiator of the session.
- `Resp`[sID] $\in [\mu]$ denotes the responder of the session.
- `Type`[sID] $\in$ {"In", "Re"} denotes the session's view, i. e. whether the initiator or the responder computes the session key.
- $\mathtt{Msg}_{\mathsf{I},1}$[sID] denotes the first message that was computed by the initiator.
- $\mathtt{Msg}_\mathsf{R}$[sID] denotes the message that was computed by the responder.
- $\mathtt{Msg}_{\mathsf{I},2}$[sID] denotes the final message that was computed by the initiator.
- `ST`[sID] denotes the (secret) state information, i. e. ephemeral secret keys.
- `SK`[sID] denotes the session key. If the session terminates without a valid session key, we set this variable to the special string "reject".

To establish a session between two parties, the adversary is given access to oracles $\textsc{Session}_\mathrm{I}$ and $\textsc{Session}_\mathsf{R}$, where the first one starts a session of type "In" and the second one of type "Re". In order to complete a session, oracles $\textsc{Der}_\mathrm{I}$ and $\textsc{Der}_\mathsf{R}$ have to be queried.The adversary has also access to oracles Corr, Reveal and Rev-State to obtain secret information. (The latter is only available if state reveal attacks are considered.) We use the following boolean values to keep track of which queries the adversary made:

- `cor`[n] denotes whether the long-term secret key of party P_n was given to the adversary.
- `peerPreCor`[sID] denotes whether the peer of the session was corrupted and its long-term key was given to the adversary *before* the owner's session key was computed, which is important for forward security.

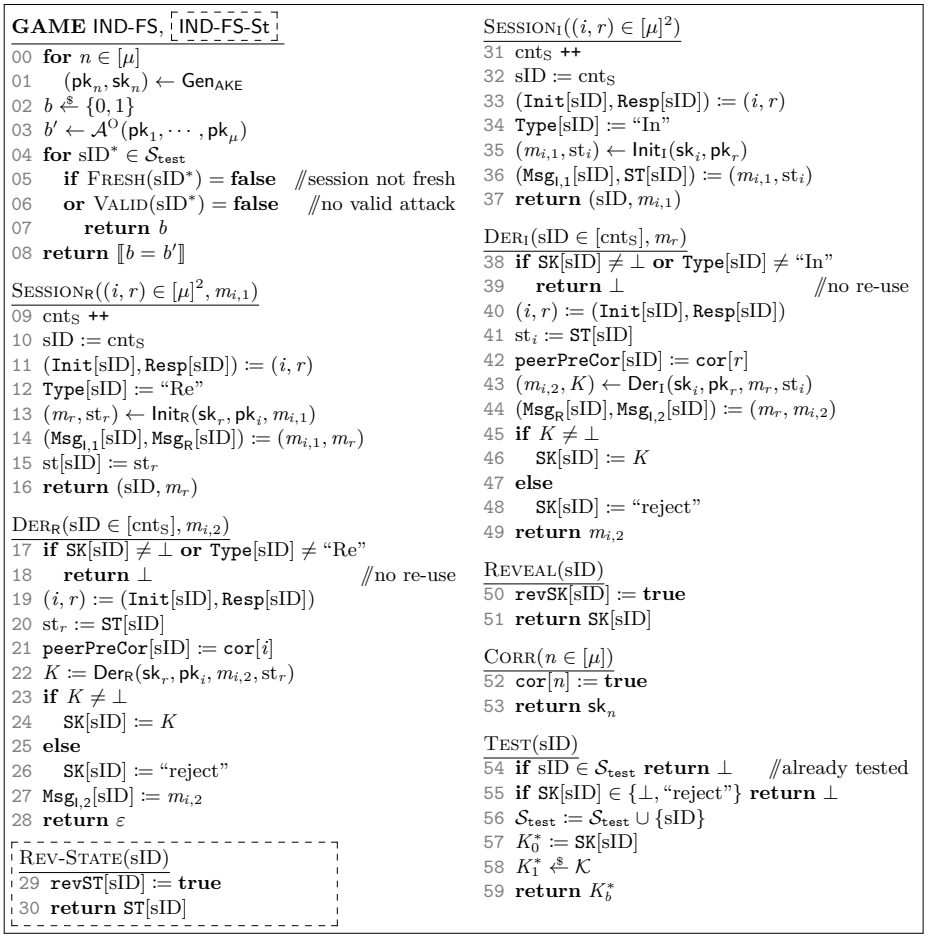

Fig. 2. Games IND-FS and IND-FS-St for AKE. Rev-State is only available in IND-FS-St. In IND-FS, $\mathcal{A}$ has access to oracles $\mathrm{O} := \{\text{Session}_{\mathrm{I}}, \text{Session}_{\mathrm{R}}, \text{Der}_{\mathrm{I}}, \text{Der}_{\mathrm{R}}, \text{Reveal}, \text{Corr}, \text{Test}\}$. In IND-FS-St, $\mathcal{A}$ has access to the oracles in IND-FS and the Rev-State oracle. Helper procedures Fresh and Valid are defined in Figure 3. If there exists any test session which is neither fresh nor valid, the game will return b.

- `revST`[sID] denotes whether the session state was given to the adversary.
- `revSK`[sID] denotes whether the session key was given to the adversary.

The adversary can forward messages between sessions or modify them. By that, we can define the relationship between two sessions:

- **Matching Session**: Two sessions sID and sID′ *match* if the same parties are involved, the messages sent and received are the same they are of different types (cf. line 14 in Figure 3).
- **Partially Matching Session**: A session sID has a *partially matching* session sID′ if the same parties are involved, the messages sent and received are the same without considering the last message and they are of different types, where sID′ is of type "Re" (cf. line 16 in Figure 3).

Finally, the adversary is given access to oracle Test which can be queried multiple times and which will return either the session key of the specified session or a uniformly random key. We use one bit b for all queries, and store test sessions in a set $\mathcal{S}_{\mathtt{test}}$. For each test session, we require that the adversary does not issue queries such that the session key can be trivially computed. In Figure 3 we define the properties of freshness and validity which all test sessions have to satisfy:

- **Freshness**: A (test) session is called *fresh* if the session key was not revealed. Furthermore, if there exists a matching session, we require that this session's key is not revealed and that this session is not also a test session.
- **Validity**: A (test) session is called *valid* if it is fresh and the adversary performed any attack which is defined in the security model. For game IND-FS-St, we capture this with attacks listed in Table 4 in Appendix B. For game IND-FS, we use Table 2 to capture valid attacks.

```
FRESH(sID*)
00 𝔐(sID*) := MATCH(sID*)
01 𝔓(sID*) := PARTIALMATCH(sID*)
02 if |𝔐(sID*)| > 1 or |𝔓(sID*)| > 1 return true
03 if revSK[sID*] or
     (∃sID ∈ 𝔐(sID*) : revSK[sID] = true)
04   return false
05 if ∃sID ∈ 𝔐(sID*) s. t. sID ∈ 𝒮test
06   return false
07 return true

VALID(sID*)
08 𝔐(sID*) := MATCH(sID*)
09 𝔓(sID*) := PARTIALMATCH(sID*)
10 if |𝔐(sID*)| > 1 or |𝔓(sID*)| ≥ 1 return true
11 for attack ∈ Table 2 and attack ∈ Table 4
12   if attack = true return true
13 return false

MATCH(sID*)                                                    //matching sessions
14 𝔐(sID*) := {sID | (Init[sID], Resp[sID]) = (Init[sID*], Resp[sID*]) ∧ (Msg_{I,1}[sID], Msg_R[sID],
                Msg_{I,2}[sID]) = (Msg_{I,1}[sID*], Msg_R[sID*], Msg_{I,2}[sID*]) ∧ Type[sID] ≠ Type[sID*]}
15 return 𝔐(sID*)

PARTIALMATCH(sID*)                                             //partially matching sessions
16 𝔓(sID*) := {sID | (Init[sID], Resp[sID]) = (Init[sID*], Resp[sID*]) ∧ (Msg_{I,1}[sID], Msg_R[sID]) =
                (Msg_{I,1}[sID*], Msg_R[sID*]) ∧ Type[sID] ≠ Type[sID*] ∧ Type[sID] = "Re"}
17 return 𝔓(sID*)
```

Fig. 3. Helper procedures FRESH and VALID for games IND-FS and IND-FS-St defined in Figure 2. Procedure FRESH checks if the adversary performed some trivial attack. In procedure VALID, each attack is evaluated by the set of variables shown in Table 2 (for game IND-FS) or Table 4 (for game IND-FS-St) and checks if an allowed attack was performed, where the latter includes session-state reveal attacks. If the values of the variables are set as in the corresponding row, the attack was performed, i. e. attack = **true**, and thus the session is valid.

$\mathcal{A}$ gets (Initiator, Responder)	$\mathtt{cor}[i^*]$	$\mathtt{cor}[r^*]$	$\mathtt{peerPreCor}[\mathrm{sID}^*]$	$\mathtt{Type}[\mathrm{sID}^*]$	$\lvert\mathfrak{M}(\mathrm{sID}^*)\rvert$	$\lvert\mathfrak{P}(\mathrm{sID}^*)\rvert$
1. **(long-term, long-term)**	–	–	–	"In"	–	1
2. **(long-term, long-term)**	–	–	–	"Re"	1	–
5. **(long-term, long-term)**	–	–	**F**	"In"	–	0
6. **(long-term, long-term)**	–	–	**F**	"Re"	0	–

Table 2. Table of attacks for adversaries against three-message protocols with FS. An attack is regarded as an AND conjunction of variables with specified values as shown in the each line, where "–" means that this variable can take arbitrary value and **F** means "false". This table is obtained from Table 3 by excluding all trivial attacks.

If the protocol does not use appropriate randomness, it should not be considered secure. In this case, there can be multiple matching sessions to a test session, which an adversary can take advantage of. We capture this as part of the validity property (cf. line 10). For an honest run of the protocol, the underlying min-entropy ensures that this attack will only happen with negligible probability.

We define validity of different attack strategies in Table 2, using variables to indicate which queries the adversary may (not) make. The purpose is to make our proofs precise by listing all the possible and non-trivial attacks. Attacks covered in the IND-FS model capture *forward secrecy* (FS) and *key compromise impersonation* (KCI) attacks. We provide a more detailed description of Table 2 and the full table for IND-FS-St in Appendix B. For all test sessions, at least one attack has to evaluate to true. Then, the adversary wins if he distinguishes the session keys from uniformly random keys which he obtains through queries to the TEST oracle.

Definition 2 (Key Indistinguishability of AKE). *We define games* IND-FS *and* IND-FS-St *as in Figures 2 and 3. We say* AKE *is* $(t, \varepsilon, \mu, S, T, Q_{\mathrm{COR}})$-IND-FS*-secure resp.* $(t', \varepsilon', \mu, S, T, Q_{\mathrm{COR}}, Q_{\mathrm{ST}})$-IND-FS-St*-secure if for all adversaries* $\mathcal{A}$ *attacking the protocol in time* t *resp.* t' *with* μ *users,* S *sessions,* T *test queries,* Q_{COR} *corruptions, and* Q_{ST} *state reveals, we have*

$$\left|\Pr[\mathsf{IND\text{-}FS}^{\mathcal{A}}_{\mathsf{AKE}} \Rightarrow 1] - \frac{1}{2}\right| \leq \varepsilon \quad \textit{resp.} \quad \left|\Pr[\mathsf{IND\text{-}FS\text{-}St}^{\mathcal{A}}_{\mathsf{AKE}} \Rightarrow 1] - \frac{1}{2}\right| \leq \varepsilon' \ .$$

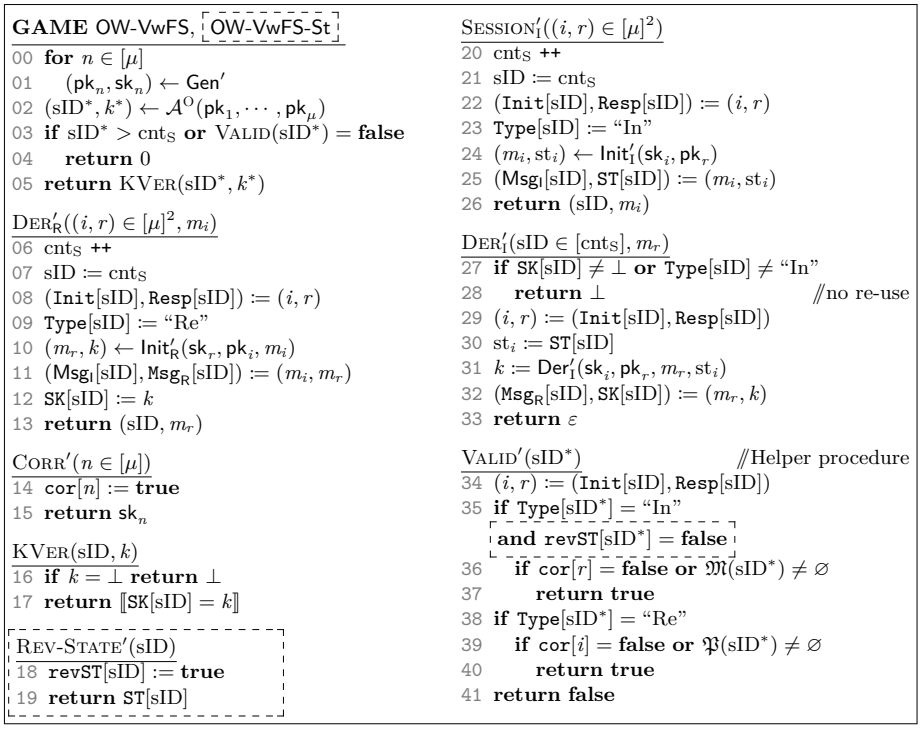

Fig. 4. Games OW-VwFS (without dashed boxes) and OW-VwFS-St (including dashed boxes) for AKE'. $\mathcal{A}$ has access to oracles $\mathrm{O} \coloneqq \{\textsc{Session}'_\mathrm{I}, \textsc{Der}'_\mathsf{R}, \textsc{Der}'_\mathrm{I}, \textsc{Corr}', \textsc{KVer}\}$. In OW-VwFS-St, $\mathcal{A}$ also has access to $\textsc{Rev-State}'$. In two-message AKE, responder sessions do not have state. So, $\textsc{Rev-State}'(\mathrm{sID})$ will return $\bot$ if sID is a responder session. Further, partially matching session is defined as $\mathfrak{P}(\mathrm{sID}^*) \coloneqq \{\mathrm{sID} \mid \mathtt{Type}[\mathrm{sID}] = \text{"In"} \wedge (\mathtt{Init}[\mathrm{sID}], \mathtt{Resp}[\mathrm{sID}]) = (\mathtt{Init}[\mathrm{sID}^*], \mathtt{Resp}[\mathrm{sID}^*]) \wedge \mathsf{Msg}_\mathsf{I}[\mathrm{sID}] = \mathsf{Msg}_\mathsf{I}[\mathrm{sID}^*]\}$.

Note that if there exists a session which is neither fresh nor valid, the game outputs the bit b, which implies that $\Pr[\mathsf{IND\text{-}FS}^{\mathcal{A}}_{\mathsf{AKE}} \Rightarrow 1] = \frac{1}{2}$ or $\Pr[\mathsf{IND\text{-}FS\text{-}St}^{\mathcal{A}}_{\mathsf{AKE}} \Rightarrow 1] = \frac{1}{2}$, giving the adversary an advantage equal to 0. This captures that an adversary will not gain any advantage by performing a trivial attack.

4 Verifiable Authenticated Key Exchange

To build a tightly secure three-message AKE protocol with key confirmation from a two-message AKE protocol, we define two security notions of the two-message protocol: The first one is One-Way against key Verification attacks and weak Forward Secrecy, or OW-VwFS for short, and the second one is OW-VwFS with state-reveal attacks, or OW-VwFS-St for short.

We define the syntax of a two-message key exchange protocol in a similar fashion as the three-message AKE. Let $\mathsf{AKE}' \coloneqq (\mathsf{Setup}', \mathsf{Gen}', \mathsf{Init}'_\mathrm{I}, \mathsf{Init}'_\mathsf{R}, \mathsf{Der}'_\mathrm{I})$, where Setup', Gen' and $\mathsf{Init}'_\mathrm{I}$ are defined exactly as in the three-message protocol. instead of a state, the responder's algorithm $\mathsf{Init}'_\mathsf{R}$ computes a session key K. The initiator's algorithm Der'_I does not output a second message, but only the session key. Correctness is defined similarly to the three-message case.

Definition 3 (Correctness of two-message AKE). *Let* $\mathsf{AKE}' \coloneqq (\mathsf{Setup}', \mathsf{Gen}', \mathsf{Init}'_\mathrm{I}, \mathsf{Init}'_\mathsf{R}, \mathsf{Der}'_\mathrm{I})$ *be an AKE protocol. We say* AKE' *is* ρ*-correct if*

$$\Pr\left[K_i = K_r \neq \bot : \begin{array}{l} \mathsf{par}' \leftarrow \mathsf{Setup}'(1^\lambda), \\ (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}'(\mathsf{par}), (\mathsf{pk}_r, \mathsf{sk}_r) \leftarrow \mathsf{Gen}'(\mathsf{par}), \\ (m_i, \mathrm{st}_i) \leftarrow \mathsf{Init}'_\mathrm{I}(\mathsf{sk}_i, \mathsf{pk}_r), \\ (m_r, K_r) \leftarrow \mathsf{Init}'_\mathsf{R}(\mathsf{sk}_r, \mathsf{pk}_i, m_i), \\ K_i \coloneqq \mathsf{Der}'_\mathrm{I}(\mathsf{sk}_i, \mathsf{pk}_r, m_r, \mathrm{st}_i) \end{array}\right] \geq \rho,$$

where the probability is taken over the randomness of $\mathsf{Setup}', \mathsf{Gen}', \mathsf{Init}'_\mathrm{I}$, *and* $\mathsf{Init}'_\mathsf{R}$.

OW-VwFS is similar to the standard weak forward secrecy, but an adversary is additionally allowed to check if a key corresponds to some generated transcripts. The security notion OW-VwFS-St, based on OW-VwFS, allows the adversary to reveal session states. Moreover, these two security notions do not have Reveal and Test oracles. Our notion is motivated by the one-wayness against honest and key verification attacks in [PQR22], but it is stronger in the sense that it allows active attacks. These are formally defined by Definition 4 with security games OW-VwFS and OW-VwFS-St as in Figure 4.

Definition 4 (OW-VwFS and OW-VwFS-St security). *A two-message authenticated key exchange protocol* AKE' *is* $(t, \varepsilon, \mu, S, Q_{\text{COR}}, Q_{Ver})$-OW-VwFS *secure resp.* $(t', \varepsilon', \mu, S, Q_{\text{COR}}, Q_{Ver}, Q_{ST})$-OW-VwFS-St *secure, where* μ *is the number of users,* S *is the number of sessions,* Q_{Ver} *is the number of calls to* KVer *and* Q_{ST} *is the number of calls to* Rev-State$'$*, if for all adversaries* $\mathcal{A}$ *attacking the protocol in time at most* t *resp.* t'*, we have*

$$\Pr[\mathsf{OW\text{-}VwFS}^{\mathcal{A}}_{\mathsf{AKE}'} \Rightarrow 1] \leq \varepsilon \quad \textit{resp.} \quad \Pr[\mathsf{OW\text{-}VwFS\text{-}St}^{\mathcal{A}}_{\mathsf{AKE}'} \Rightarrow 1] \leq \varepsilon' \ .$$

Valid attacks are defined via Valid$'$. For the session sID* for which the adversary aims to compute the session key, we basically allow two types of attacks: If there is a (partially) matching session, then both parties may be corrupted. Otherwise, the adversary must not corrupt the peer of the session. Additionally for the model with state reveal attacks, the state for sID* must not be revealed in any case.

Min-Entropy. We require that public keys have γ *bits of min-entropy*, i. e., for all $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{Gen}'$, $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{Gen}'$, we have $\Pr[\mathsf{pk}_0 = \mathsf{pk}_1] \leq 2^{-\gamma}$. Similarly, we require that messages have α *bits of min-entropy*, i. e., for all messages m' we have $\Pr[m = m'] \leq 2^{-\alpha}$, where m is output by either $\mathsf{Init}'_{\mathrm{I}}$ or $\mathsf{Init}'_{\mathrm{R}}$.

5 AKE with Key Confirmation

We now build a three-message AKE protocol $\mathsf{AKE}_{\mathsf{KC}}$ with key confirmation from a two-message AKE protocol AKE' and three hash functions $\mathsf{G}_{\mathrm{I}}, \mathsf{G}_{\mathrm{R}}, \mathsf{H}$. An overview is given in Figure 5. Hash functions G_{I}, G_{R} and H are defined as follows: $\mathsf{G}_{\mathrm{I}}, \mathsf{G}_{\mathrm{R}} : \{0,1\}^* \to \{0,1\}^\lambda$ and $\mathsf{H} : \{0,1\}^* \to \mathcal{K}$, where λ is the length of key confirmation tags and $\mathcal{K}$ is the key space of $\mathsf{AKE}_{\mathsf{KC}}$.[6]

Let $\mathsf{AKE}' = (\mathsf{Setup}', \mathsf{Gen}', \mathsf{Init}'_{\mathrm{I}}, \mathsf{Init}'_{\mathrm{R}}, \mathsf{Der}'_{\mathrm{I}})$. We define $\mathsf{AKE}_{\mathsf{KC}}$ as follows: Setup, $\mathsf{Gen}_{\mathsf{AKE}}$, $\mathsf{Init}_{\mathrm{I}}$ will be the same as Setup', Gen' and $\mathsf{Init}'_{\mathrm{I}}$, respectively. $\mathsf{Init}_{\mathrm{R}}$ first runs $\mathsf{Init}'_{\mathrm{R}}$ to obtain the responder's message m_r and the key k of AKE', where the latter is used to derive the final session key and key confirmation messages. In particular, the responder first computes the key confirmation tag $\pi_r := \mathsf{G}_{\mathrm{R}}(k, \mathrm{ctxt})$, where ctxt is defined as the two parties' public keys and the initial messages (cf. Figure 5). It then also computes the expected key confirmation tag π'_i and session key K' using G_{I} and H on the same input. It sends (m_r, π_r) to the initiator and keeps (π'_i, K') as state. The initiator runs $\mathsf{Der}_{\mathrm{I}}$ which is defined as follows: First, it runs $\mathsf{Der}'_{\mathrm{I}}$ to get k and then performs the same computations as the responder to compute key confirmation tags π_i, π'_r and the final session key K. It accepts K if $\pi'_r = \pi_r$ and sends π_i as the final message. The responder's derivation algorithm $\mathsf{Der}_{\mathrm{R}}$ checks whether the key confirmation tag is valid, i. e., $\pi_i = \pi'_i$, and if this is the case it sets the session key to K'.

Whenever an equality check fails or the underlying algorithms of AKE' return $\perp$, the parties terminate the session, i. e., they *reject*, and return $\perp$.

Correctness. The correctness of $\mathsf{AKE}_{\mathsf{KC}}$ follows directly from the correctness of AKE'. In particular, if AKE' is $(1-\delta)$-correct, then so is $\mathsf{AKE}_{\mathsf{KC}}$.

Security. We prove IND-FS security of $\mathsf{AKE}_{\mathsf{KC}}$ based on OW-VwFS security of AKE' and modeling G_{I}, G_{R} and H as random oracles.

Theorem 1. *Let* AKE' *be* $(1-\delta)$*-correct and have public keys with* γ *bits of entropy and messages with* α *bits of entropy. Let* $\mathsf{AKE}_{\mathsf{KC}}$ *be as defined in Figure 5, where* $\mathsf{G}_{\mathrm{I}}, \mathsf{G}_{\mathrm{R}} : \{0,1\}^* \to \{0,1\}^\lambda$ *and* $\mathsf{H} : \{0,1\}^* \to$

[6] We define three different hash functions here which allows us to model them as independent random oracles. When instantiating the hash functions with the same function, one would need to use appropriate domain separation.

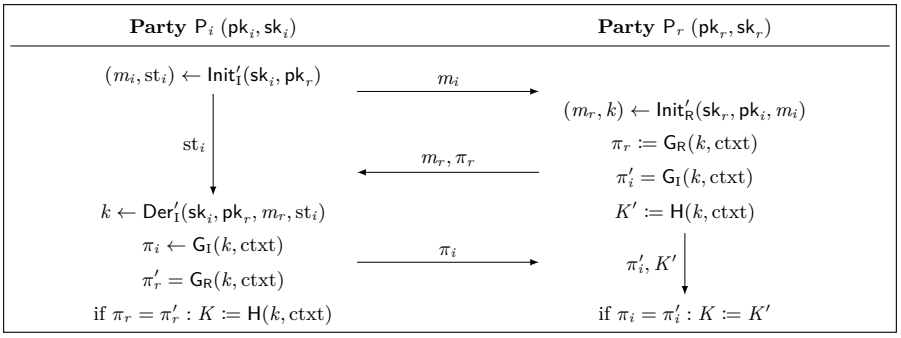

Fig. 5. AKE protocol AKE_KC from AKE' and key confirmation. The context is defined as $\mathrm{ctxt} \coloneqq (\mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r)$. G_I, G_R and H are independent random oracles.

$\mathcal{K}$ are modelled as random oracles. For every adversary $\mathcal{A}$ that breaks the $(t, \varepsilon, \mu, S, T, Q_{COR})$-IND-FS-security of AKE_KC, there exists an adversary $\mathcal{B}$ that breaks the $(t', \varepsilon', \mu, S, Q_{COR}, Q_{Ver})$-OW-VwFS security of AKE' with $t' \approx t$ and

$$\varepsilon \leq \varepsilon' + 2S \cdot \delta + (S + S^2) \cdot 2^{-\lambda} + \mu^2 \cdot 2^{-\gamma} + S(S + Q_{\mathsf{G}_\mathsf{I}} + Q_{\mathsf{G}_\mathsf{R}} + Q_\mathsf{H}) \cdot 2^{-\alpha} \ ,$$

where $Q_{\mathsf{G}_\mathsf{I}}$, $Q_{\mathsf{G}_\mathsf{R}}$ and Q_H are the number of queries to random oracles G_I, G_R and H and $Q_{Ver} \leq S + Q_{\mathsf{G}_\mathsf{I}} + Q_{\mathsf{G}_\mathsf{R}} + Q_\mathsf{H}$.

The idea of the proof is that we can simulate the key confirmation tags and session keys without knowing the key k of the underlying two-message protocol as long as it has not been queried to (one of) the random oracles. For this we have to keep track of whether the adversary trivially knows k because the session is not fresh anymore. We can handle this case and still simulate correctly using KVER oracle. The only way to win the game is to compute k for a fresh and valid session, thus breaking one-wayness of the underlying protocol. We now prove the theorem formally.

Proof. Let $\mathcal{A}$ be an adversary against IND-FS security of AKE_KC. We consider the sequence of games G_0-G_3 in Figures 6 and 7.

GAME G_0. The first game G_0 is the original IND-FS security game, however we exclude that public keys or messages collide (which means that if such events happen, then the game will abort and return a random bit). This also includes the key confirmation tags. Thus we get

$$\Pr[\mathsf{G}_0^\mathcal{A} \Rightarrow 1] \leq \Pr[\text{IND-FS}^\mathcal{A}_{\mathsf{AKE}_\mathsf{KC}} \Rightarrow 1] + \mu^2 \cdot 2^{-\gamma} + S^2 \cdot 2^{-\alpha} + S^2 \cdot 2^{-\lambda} \ .$$

Note that this means there can be at most one (partially) matching session for each session.

GAME G_1. In game G_1, we want to ensure that π_r, π_i and K have not been queried to the respective random oracle before they are determined. Note that all three values will be determined in $\textsc{Session}_\mathsf{R}$ when m_r and k are computed. Thus, whenever $\textsc{Session}_\mathsf{R}$ is queried, we check whether there already exists a query $(k', \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r)$ to G_R, G_I or H for some k' (line 19). If this is the case, we raise flag BadEntropy and abort. If BadEntropy is not raised, we draw fresh values for π_r, π_i and K and explicitly assign them to the corresponding entry of the respective random oracle (lines 21-24). We make this explicit here to prepare for the next step where we have to do a case distinction. Note that G_0 and G_1 are the same, except if BadEntropy is raised. Thus,

$$|\Pr[\mathsf{G}_0^\mathcal{A} \Rightarrow 1] - \Pr[\mathsf{G}_1^\mathcal{A} \Rightarrow 1]| \leq \Pr[\mathsf{BadEntropy}] \leq S(Q_{\mathsf{G}_\mathsf{R}} + Q_{\mathsf{G}_\mathsf{I}} + Q_\mathsf{H}) \cdot 2^{-\alpha} \ ,$$

where we bound the event by the entropy of AKE'. The message m_r is computed by the game directly before we check for this event. We then use the union bound over the maximum number of sessions S.

GAME G_2. In game G_2, we want to compute π_r, π_i and K without using k explicitly and prepare for the reduction to OW-VwFS. For this, we have to make a distinction between fresh and non-fresh sessions. We add two additional variables $\mathtt{ctxt}$[sID] and $\mathtt{k}$[sID] for each session which store the context and the session key of the underlying AKE'. When $\textsc{Session}_\mathsf{R}$ is queried, we no longer assign the random oracle entries $[k, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r]$. Instead, we use a special placeholder symbol for the key k. In particular, if the

Fig. 6. Games G_0-G_3 for the proof of Theorem 1. $\mathcal{A}$ has access to oracles $\mathrm{O} \coloneqq \{\textsc{Session}_{\mathrm{I}}, \textsc{Session}_{\mathsf{R}}, \textsc{Der}_{\mathrm{I}}, \textsc{Der}_{\mathsf{R}}, \textsc{Reveal}, \textsc{Corr}, \textsc{Test}, \mathsf{G}_{\mathrm{I}}, \mathsf{G}_{\mathsf{R}}, \mathsf{H}\}$. Helper procedures Fresh and Valid are defined in Figure 3.

session is still fresh and valid (i. e., there exists a session with a matching context up to this point, or the intended peer is not (yet) corrupted), we use the symbol $\diamond$ (lines 28-30). Otherwise, in case the session is not valid, we use the symbol $\oplus$ (lines 32-34). This distinction will be necessary to patch the random oracle correctly. Note that an adversary might be able to compute the correct key k for a non-valid session.

We describe how the random oracles are patched below. First, we explain how to change $\mathsf{Der}_{\mathrm{I}}$ and $\mathsf{Der}_{\mathsf{R}}$ accordingly. For each query to $\mathsf{Der}_{\mathrm{I}}$, we first update the context with the message m_r that was used to query the oracle. Then we check whether there exists a potential partnered session with the same context (line 61). In this case we know the corresponding values π_r, π_i and K which are stored with the symbol $\diamond$. We check whether the tag π_r is correct (if not, the session rejects) and assign the session key (lines 62-66). If there is no other session with the same context, we have to make another case distinction. In the first case, we use k explicitly to check whether there has been a query to the random oracle G_{R} such that the tag π_r matches (line 68). In this case, we proceed normally. Looking ahead, this will be a critical point in the next game modification. If there exists no such query to G_{R}, then the game makes the query and chooses a tag uniformly at random (line 74). If this tag is the same as the one provided by the adversary, we raise flag RandKC and abort (line 76). Otherwise, the session simply rejects. We modify $\textsc{Der}_{\mathsf{R}}$ in the exact same way, except that we are now looking at π_i (Figure 7, lines 09-24).

```
DER_R(sID ∈ [cnt_S], π_i)
00 if SK[sID] ≠ ⊥ or Type[sID] ≠ "Re"
01   return ⊥
02 (i, r) := (Init[sID], Resp[sID])
03 (π'_i, K') := ST[sID]
04 peerPreCor[sID] := cor[i]
05 if π_i ≠ π'_i                                   // G0-G1
06   SK[sID] := "reject"                           // G0-G1
07   return ⊥                                      // G0-G1
08 K := K'                                         // G0-G1
09 if ∃sID' s.t. ctxt[sID'] = ctxt[sID]            // G2-G3
10   if π_i ≠ G_I[⋄, pk_i, pk_r, m_i, m_r]         // G2-G3
11     SK[sID] := "reject"                         // G2-G3
12     return ⊥                                    // G2-G3
13   K := H[⋄, pk_i, pk_r, m_i, m_r]               // G2-G3
14 else                                            // G2-G3
15   if G_I[k, pk_i, pk_r, m_i, m_r] = π_i         // G2-G3
16     if cor[i] = false                           // G3
17       QueryRO := true; abort                    // G3
18     K := H(k, pk_i, pk_r, m_i, m_r)             // G2-G3
19   else                                          // G2-G3
20     G_I[⋄, pk_i, pk_r, m_i, m_r] ←$ {0,1}^λ     // G2-G3
21     if π_i = G_I[⋄, pk_i, pk_r, m_i, m_r]       // G2-G3
22       RandKC := true; abort                     // G2-G3
23     SK[sID] := "reject"                         // G2-G3
24     return ⊥                                    // G2-G3
25 (Msg_{i,2}[sID], SK[sID]) := (π_i, K)
26 return ε

G_R(k, pk_i, pk_r, m_i, m_r)
27 if G_R[⋄, pk_i, pk_r, m_i, m_r] = π ≠ ⊥          // G2-G3
28   S := {sID | ctxt[sID] = (pk_i, pk_r, m_i, m_r)} // G2-G3
29   for sID ∈ S // note |S| ≤ 2                    // G2-G3
30     if k[sID] = k                               // G2-G3
31       QueryRO := true; abort                    // G3
32       return π                                  // G2-G3
33 elseif G_R[⊕, pk_i, pk_r, m_i, m_r] = π ≠ ⊥      // G2-G3
34   Find sID s.t. ctxt[sID] = (pk_i, pk_r, m_i, m_r) // G2-G3
35   if k[sID] = k                                 // G2-G3
36     Replace ⊕ with k                            // G2-G3
37     return π                                    // G2-G3
38 if G_R[k, pk_i, pk_r, m_i, m_r] = π ≠ ⊥
39   return π
40 π ←$ {0,1}^λ
41 G_R[k, pk_i, pk_r, m_i, m_r] := π
42 return π

CORR(n ∈ [μ])
43 cor[n] := true
44 return sk_n

TEST(sID)
45 if sID ∈ S_test return ⊥
46 if SK[sID] ∈ {⊥, "reject"} return ⊥
47 S_test := S_test ∪ {sID}
48 K*_0 := SK[sID]
49 K*_1 ←$ K
50 return K*_b
```

Fig. 7. Oracles for games G_0-G_3 for the proof of Theorem 1. G_I and H are defined analogously to G_R.

Before bounding RandKC, we explain the simulation of random oracles as described in Figure 7. We explain G_R in more detail. (G_I and H are modeled in exactly the same way.) For each query $(k, \mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r)$, we first check for entries with the special symbol. More specifically, if there exists an entry with the given context and the symbol $\diamond$, we look for the sID with this context. Note that there can be at most two sessions (one of type "In" and one of type "Re" which will be matching sessions), which we capture by computing a set $\mathcal{S}$ containing the corresponding sID(s) (line 29). If the key k of the random oracle query corresponds to the one stored in $\mathtt{k}$[sID][7], then G_R simply outputs the stored value π (line 32). We do the same for special symbol $\oplus$ (line 34, note that here sID is always unique), except that we also update the entry accordingly, i. e., replace $\oplus$ with k if $k = \mathtt{k}$[sID] (line 36). This way, the simulation is consistent with the other oracles.

Overall, the two games only differ when flag RandKC is raised. Note that we can bound the probability that a tag is valid without the random oracle being queried by the length of the tag. Union bound over S session gives us

$$|\Pr[\mathsf{G}_1^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathsf{G}_2^{\mathcal{A}} \Rightarrow 1]| \leq \Pr[\mathsf{RandKC}] \leq S \cdot 2^{-\lambda} + S \cdot \delta \ .$$

Game G_3. In the final game G_3, we raise flag QueryRO if the adversary ever queries the random oracle on a key k of a fresh session. Depending on the order of queries, this event can occur for different oracles: DER_I (Figure 6, line 70), DER_R (Figure 7, line 17) or one of the random oracles (Figure 7, line 31). First, we look at sessions that do not have a matching session with the same context. For queries to oracle DER_I we check the validity of π_r in line 68. If the peer r is not corrupted, then the session is still fresh and valid. Thus, we raise QueryRO if there has been a query to G_R on the correct key and context such that the output is indeed π_r. We proceed similarly for responder sessions when DER_R is queried, checking for queries to G_I. This means that all sessions where the peer is uncorrupted and no session with a matching context exist will reject (or abort).

We now look at sessions that have a session with matching context and whose relevant random oracle entries are marked with $\diamond$. Whenever one of the random oracles is queried, we check whether the key matches the one stored in $\mathtt{k}$[sID] (as described earlier) and if this is the case, we also raise QueryRO and abort.

[7] Since we cannot check correctness efficiently in the reduction which we will build in the next step, we explicitly perform the test here for all sessions in $\mathcal{S}$. However, if $\mathcal{S}$ indeed contains two sessions, then by correctness, this key (and thus the outcome) will be the same.

We claim that

$$|\Pr[\mathsf{G}_2^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathsf{G}_3^{\mathcal{A}} \Rightarrow 1]| \leq \Pr[\mathsf{QueryRO}] \leq \varepsilon' + S \cdot \delta \ .$$

Before proving the claim, note that in G_3 we have $\Pr[\mathsf{G}_3^{\mathcal{A}} \Rightarrow 1] = 1/2$. For this, observe that all sessions must have a (partially) matching session and that random oracle H is never queried on k for any of those sessions. Thus, the session key is indistinguishable from a uniformly random key.

Bounding event QueryRO. We now describe an adversary $\mathcal{B}$ against OW-VwFS security of the underlying AKE' to bound event QueryRO. A pseudocode description is given in Figure 8. The idea is that whenever $\mathcal{A}$ queries one of the random oracles on the underlying key k of a fresh and valid session (either in order to forge a key confirmation tag or to distinguish the actual session key), we can use this to break OW-VwFS security of AKE', where the verification oracle KVer is used to simulate the random oracles consistently.

We now describe $\mathcal{B}$ in more detail. It gets as input μ public keys and forwards them to $\mathcal{A}$. $\mathcal{B}$ simulates queries to oracle $\textsc{Session}_{\mathsf{I}}$ in a straightforward way by querying its own oracle $\textsc{Session}'_{\mathsf{I}}$ which returns (sID, m_i). After assigning the corresponding variables, $\mathcal{B}$ forwards the output to $\mathcal{A}$. Queries to $\textsc{Session}_{\mathsf{R}}$ are simulated as in game G_3. $\mathcal{B}$ first queries $\textsc{Der}'_{\mathsf{R}}$ to receive (sID, m_r). Instead of checking whether $k = \bot$, $\mathcal{B}$ checks whether $m_r = \bot$. If this is the case, it rejects and outputs $\bot$. Otherwise, it proceeds as described in G_3, preparing random oracle assignments by assigning fresh values to π_i, π_r and K and returning $(\mathrm{sID}, (m_r, \pi_r))$.

When $\mathcal{A}$ queries $\textsc{Der}_{\mathsf{I}}$, $\mathcal{B}$ queries $\textsc{Der}'_{\mathsf{I}}$. $\mathcal{B}$ will not be able to explicitly check whether the session key was computed successfully, however, we will argue that the simulation is consistent by correctness of AKE' and the validity of the key confirmation tag. Thus, $\mathcal{B}$ directly proceeds as described in G_3. Whenever there exists a session with the same context, then the key confirmation tag must be the same as the one computed by that session, up to correctness of AKE'. Thus the simulation is perfect except with probability $S \cdot \delta$. Whenever there exists no (partially) matching session, $\mathcal{B}$ needs to check whether G_{R} was already queried on the correct k. For this it checks all random oracle queries that have output π_r provided by $\mathcal{A}$. If such a query exists, it will be unique since we excluded collisions in the first game. $\mathcal{B}$ checks whether the respective key of the query is the correct key using its oracle KVer. If this is the case, we further distinguish two cases, based on whether the session still qualifies for a valid test session or not. If the peer of the session has not been corrupted yet, then this is a valid session and $\mathcal{B}$ outputs (sID, k) as solution in its own game. Otherwise, it proceeds. Oracle $\textsc{Der}_{\mathsf{R}}$ is simulated similarly, looking at G_{I} instead of G_{R}.

Oracle Reveal and Corr can be simulated in a straightforward way. The latter requires $\mathcal{B}$ to query its own oracle $\textsc{Corr}'$. Queries to Test will always return the real session key. Note that this is a perfect simulation since session keys are perfectly hidden unless QueryRO happens in which case $\mathcal{B}$ stops because it breaks OW-VwFS security.

It remains to describe the simulation of random oracles. The simulation works similar for all three oracles and we will describe oracle G_{R} here. As in G_3, $\mathcal{B}$ first checks whether there exists an entry with the special symbol $\diamond$. If this is the case, it finds the corresponding sID and uses the KVer oracle to check whether the key k provided by $\mathcal{A}$ belongs to this session. Since $\diamond$ is used to mark sessions that have a (partially) matching session, $\mathcal{B}$ can always use this key to win the OW-VwFS game. If there is no entry with $\diamond$ but one with $\oplus$, $\mathcal{B}$ again queries the KVer oracle, but this time it updates the corresponding entry with the correct key (if KVer returns **true**). This way, $\mathcal{B}$ can perfectly simulate non-test sessions. If none of these cases happen or KVer has returned **false**, then $\mathcal{B}$ proceeds as usual by lazy sampling.

This concludes the description of $\mathcal{B}$. Note that if QueryRO happens in game G_3, i. e., there exists a random oracle query for a fresh and valid session with correct key k, then $\mathcal{B}$ wins game OW-VwFS. We get $\Pr[\mathsf{QueryRO}] \leq \varepsilon' + S \cdot \delta$.

Further, note that $\mathcal{B}$ issues at most $(S + Q_{\mathsf{G}_{\mathsf{I}}} + Q_{\mathsf{G}_{\mathsf{R}}} + Q_{\mathsf{H}})$ to KVer since we have excluded collisions of tags in the first game. The number of queries to all other oracles is preserved. This completes the proof of Theorem 1.

AKE with Key Confirmation against State Reveal. Based on $\mathsf{AKE}_{\mathsf{KC}}$, we build a three-message AKE protocol $\mathsf{AKE}_{\mathsf{stKC}}$ that is secure against state-reveal attacks (cf. Definition 2). Since $\mathsf{AKE}_{\mathsf{stKC}}$ has a similar structure with $\mathsf{AKE}_{\mathsf{KC}}$, we follow the notations used in defining $\mathsf{AKE}_{\mathsf{KC}}$ (cf. Figure 5). An overview of $\mathsf{AKE}_{\mathsf{stKC}}$ is given in Figure 9.

$\mathsf{AKE}_{\mathsf{stKC}}$ uses the state-encryption technique [JKRS21,PWZ23a] to protect session states. Concretely, let $\mathsf{G}_{\mathsf{stI}} : \{0,1\}^{\kappa} \times \{0,1\}^{\kappa} \to \{0,1\}^{d_{\mathsf{I}}}$ and $\mathsf{G}_{\mathsf{stR}} : \{0,1\}^{\kappa} \times \{0,1\}^{\kappa} \to \{0,1\}^{d_{\mathsf{R}}}$ be two hash functions.

Fig. 8. Adversary $\mathcal{B}$ against $\mathsf{OW\text{-}VwFS}$. $\mathcal{A}$ has access to oracles $\text{O} := \{\text{SESSION}_{\text{I}}, \text{SESSION}_{\text{R}}, \text{DER}_{\text{I}}, \text{DER}_{\text{R}}, \text{REVEAL}, \text{CORR}, \text{TEST}, \mathsf{G}_{\text{I}}, \mathsf{G}_{\text{R}}, \mathsf{H}\}$. Helper procedures FRESH and VALID are defined in Figure 3. G_{I} and H are defined analogously to G_{R}.

We assume that any initiator session state of the underlying two-message AKE protocol AKE' can be encoded as a d_{I}-bit string and $d_{\text{R}} = 2\lambda$ (the length of key confirmation tag plus the length of session key derived by AKE'). $\mathsf{AKE}_{\mathsf{stKC}}$ proceeds the same as $\mathsf{AKE}_{\mathsf{KC}}$ except that (1) the long-term secret key of user i in $\mathsf{AKE}_{\mathsf{stKC}}$ also include a uniformly random key $\mathsf{s}_i \in \{0,1\}^\kappa$, and (2) each session will sample a one-time key IV uniformly at random and encrypt the session state of $\mathsf{AKE}_{\mathsf{KC}}$ via XORing with the one-time pad $\mathsf{G}_{\mathsf{stI}}(\mathsf{s}_i, \mathsf{IV})$. Now the session state (that the adversary can reveal in the state-reveal AKE model) is (IV, φ). Dashed parts in Figure 9 shows how this technique works.

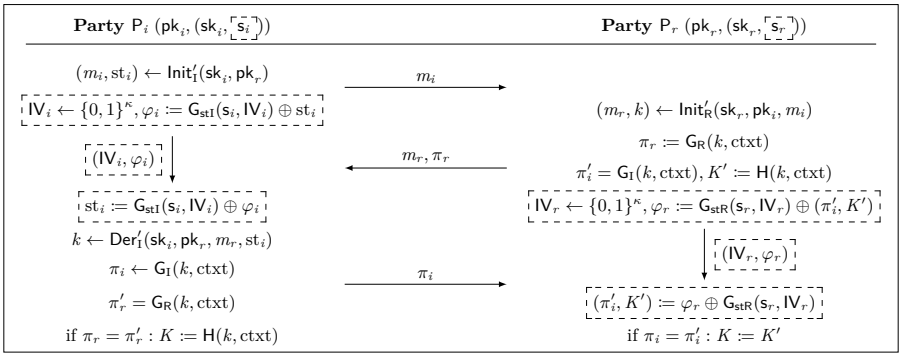

Fig. 9. AKE protocol $\mathsf{AKE}_{\mathsf{stKC}}$ from AKE', key confirmation, and state encryption. The context is defined as $\mathrm{ctxt} := (\mathsf{pk}_i, \mathsf{pk}_r, m_i, m_r)$. G_{I}, G_{R}, and H are independent random oracles. Dashed parts show how we use the state encryption technique to protect the session states. $\mathsf{G}_{\mathsf{stI}}$ and $\mathsf{G}_{\mathsf{stR}}$ are independent random oracles used for state encryption.

Correctness. Similar to $\mathsf{AKE}_{\mathsf{KC}}$, the correctness of $\mathsf{AKE}_{\mathsf{stKC}}$ follows directly from the correctness of AKE'. If AKE' is $(1-\delta)$-correct, then so is $\mathsf{AKE}_{\mathsf{stKC}}$.

Security. In Theorem 2, we prove IND-FS-St security of $\mathsf{AKE}_{\mathsf{KC}}$ based on OW-VwFS-St security of AKE' and modeling G_{I}, G_{R}, H, $\mathsf{G}_{\mathsf{stI}}$, and $\mathsf{G}_{\mathsf{stR}}$ as random oracles. Here we sketch the proof idea. By using the state encryption technique, the adversary cannot learn the unencrypted states of the underlying two-message AKE, unless it reveals the encrypted session state and corrupts the owner of the session. But this makes the session invalid and thus, it cannot be tested. Therefore, for valid sessions, state-reveal queries do not give any advantage to the adversary, and thus we can use the proof idea of Theorem 1. The full proof of Theorem 2 is postponed to Appendix C.

Theorem 2. *Let AKE' be $(1-\delta)$-correct and have public keys with γ bits of entropy and messages with α bits of entropy. Let $\mathsf{AKE}_{\mathsf{stKC}}$ be as defined in Figure 5, where $\mathsf{G}_{\mathsf{I}}, \mathsf{G}_{\mathsf{R}} : \{0,1\}^* \to \{0,1\}^\lambda$, $\mathsf{H} : \{0,1\}^* \to \mathcal{K}$, $\mathsf{G}_{\mathsf{stI}} : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^{d_{\mathsf{I}}}$, and $\mathsf{G}_{\mathsf{stR}} : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^{d_{\mathsf{R}}}$ are modeled as random oracles. For every adversary $\mathcal{A}$ that breaks the $(t, \varepsilon, \mu, S, T, Q_{COR}, Q_{ST})$-IND-FS-St-security of $\mathsf{AKE}_{\mathsf{stKC}}$, there exists an adversary $\mathcal{B}$ that breaks the $(t', \varepsilon', \mu, S, Q_{COR}, Q_{Ver}, S)$-OW-VwFS-St security of AKE' with $t' \approx t$ and*

$$\begin{aligned}\varepsilon \leq \varepsilon' + 2S \cdot \delta + (\mu^2 + S^2 + \mu Q_{\mathsf{G}_{\mathsf{stI}}} + 2SQ_{\mathsf{G}_{\mathsf{stI}}}) \cdot 2^{-\kappa} \\ + \mu^2 \cdot 2^{-\gamma} + (S + S^2) \cdot 2^{-\lambda} + (Q_{\mathsf{G}_{\mathsf{R}}} + Q_{\mathsf{G}_{\mathsf{I}}} + Q_{\mathsf{H}} + S) \cdot S \cdot 2^{-\alpha},\end{aligned}$$

where Q_{h} is the number of queries to the respective random oracle h and $Q_{Ver} \leq S + Q_{\mathsf{G}_{\mathsf{I}}} + Q_{\mathsf{G}_{\mathsf{R}}} + Q_{\mathsf{H}}$.

6 Applying our Results to Existing Protocols

We first show how to construct verifiable AKE from KEMs which gives us tight AKE with key confirmation and perfect forward secrecy from lattices and DDH. The advantage is that we do not have to consider random oracles and the proofs are comparably simpler than those of full AKE security. We then show how we can recover the optimal tightness bound for the CCGJJ protocol [CCG+19] using our modular transformation rather than that of [GGJJ23].

6.1 AKE from KEMs

We provide results for KEM-based AKE secure without and with state reveal, where the former allows for weaker assumptions. The protocol, denoted by $\mathsf{AKE}'_{\mathsf{kem}}$, to which we want to apply our compiler from the previous section is given in Figure 10. Each party holds long-term keys of a KEM scheme KEM_1 and in each session, an ephemeral key using KEM_0 is exchanged. The session key then simply consists of three KEM keys. We also denote its variant with key confirmation by $\mathsf{AKE}_{\mathsf{kem}}$ (i. e., combining Figure 10 with Figure 5) and the one resisting state reveals by $\mathsf{AKE}_{\mathsf{st,kem}}$ (i. e., combining Figure 10 with Figure 9).

One-Way Security of KEM. Depending on whether the KEM is used for long-term keys or ephemeral keys and whether state reveals are allowed, we need a different variant of one-way security: multi-user

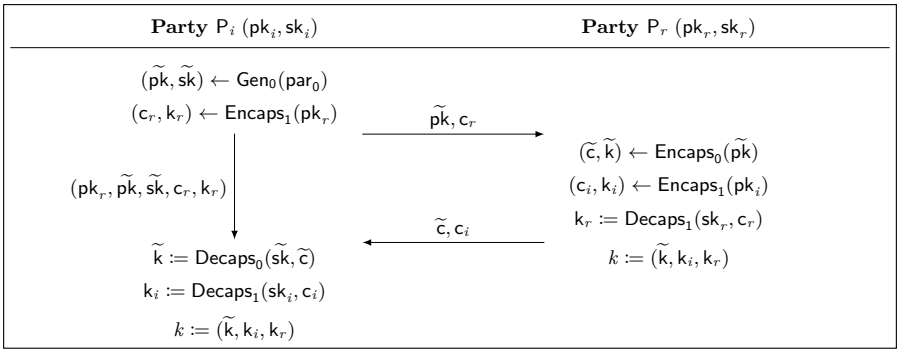

Fig. 10. AKE protocol $\mathsf{AKE}'_{\mathsf{kem}}$ from KEM schemes KEM_1, KEM_0.

one-way security under plaintext checking and ciphertext validity attacks without (OW-PCVA) or with corruptions (OW-PCVA-C), and with corruptions and reveal queries (OW-PCVA-CR). These notions are weaker variants of OW-ChCCA security from Pan, Wagner and Zeng [PWZ23a] and are tightly implied by OW-ChCCA. The formal security definitions are given in Appendix A.

Analysis of $\mathsf{AKE}'_{\mathsf{kem}}$ and $\mathsf{AKE}_{\mathsf{kem}}$. We prove that $\mathsf{AKE}'_{\mathsf{kem}}$ protocol is a secure verifiable AKE protocol. When not considering state reveals, we can use weaker assumptions, namely OW-PCVA and OW-PCVA-C. We also prove security with state-reveals which uses the definition of OW-PCVA-CR security. We then apply Theorem 1 resp. 2 to obtain $\mathsf{AKE}_{\mathsf{kem}}$ which has full forward secrecy.

Correctness and Entropy. Let KEM_0 be $(1-\delta_0)$-correct and have public keys with γ_0 bits of entropy and messages with α_0 bits of entropy. Let KEM_1 be $(1-\delta_1)$-correct and have public keys with γ_1 bits of entropy and messages with α_1 bits of entropy. Then $\mathsf{AKE}'_{\mathsf{kem}}$ is $(1-\delta_0-2\delta_1)$-correct. Further, $\mathsf{AKE}'_{\mathsf{kem}}$ has public keys with γ_1 bits of entropy and messages with at least $\min(\gamma_0, \alpha_0, \alpha_1 - 1)$ bits of entropy.

We now establish OW-VwFS and OW-VwFS-St security of $\mathsf{AKE}'_{\mathsf{kem}}$ and defer the proofs to Appendix D.1.

Lemma 1. *For every adversary $\mathcal{A}$ that breaks the $(t, \varepsilon, \mu, S, T, Q_{COR}, Q_{Ver})$-OW-VwFS security of $\mathsf{AKE}'_{\mathsf{kem}}$, there exist adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ that break $(t_1, \varepsilon_1, S, S, S, Q_{Ver})$-OW-PCVA security of KEM_0 and $(t_2, \varepsilon_2, \mu, S, S, 2Q_{Ver}, Q_{COR})$-OW-PCVA-C security of KEM_1 with $t_1 \approx t_2 \approx t$ and $\varepsilon \leq \varepsilon_1 + \varepsilon_2$.*

Lemma 2. *For every adversary $\mathcal{A}$ that breaks the $(t, \varepsilon, \mu, S, T, Q_{COR}, Q_{Ver}, Q_{ST})$-OW-VwFS-St security of $\mathsf{AKE}'_{\mathsf{kem}}$, there exist adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ that break $(t_1, \varepsilon_1, S, S, S, Q_{Ver}, Q_{ST})$-OW-PCVA-C security of KEM_0 and $(t_2, \varepsilon_2, \mu, S, S, 2Q_{Ver}, Q_{COR}, Q_{ST})$-OW-PCVA-CR security of KEM_1 with $t_1 \approx t_2 \approx t$ and $\varepsilon \leq \varepsilon_1 + \varepsilon_2$.*

We now add key confirmation to $\mathsf{AKE}'_{\mathsf{kem}}$ as described in Figure 5 resp. Figure 9. The following theorem then follows from combining Theorem 1 with Lemma 1 resp. Theorem 2 with Lemma 2.

Theorem 3. *Let KEM_0 be $(1-\delta_0)$-correct and have public keys with γ_0 bits of entropy and messages with α_0 bits of entropy. Let KEM_1 be $(1-\delta_1)$-correct and have public keys with γ_1 bits of entropy and messages with α_1 bits of entropy. Let $\mathsf{AKE}_{\mathsf{kem}}$ resp. $\mathsf{AKE}_{\mathsf{st,kem}}$ be defined as described above by combining Figure 10 with Figure 5 resp. Figure 9, where $\mathsf{G_I}, \mathsf{G_R} : \{0,1\}^* \to \{0,1\}^\lambda$, $\mathsf{H} : \{0,1\}^* \to \mathcal{K}$, $\mathsf{G_{stI}} : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^{d_\mathsf{I}}$ and $\mathsf{G_{stR}} : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^{d_\mathsf{R}}$ are modeled as random oracles. Let Q_h be the number of queries to the respective random oracle h.*

For any $\mathcal{A}$ against the $(t, \varepsilon, \mu, S, T, Q_{COR})$-IND-FS-security of $\mathsf{AKE}_{\mathsf{kem}}$, there exist adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ that break $(t_1, \varepsilon_1, S, S, S, Q_{Ver})$-OW-PCVA security of KEM_0 and $(t_2, \varepsilon_2, \mu, S, S, 2Q_{Ver}, Q_{COR})$-OW-PCVA-C security of KEM_1, where $Q_{Ver} \leq S + Q_{\mathsf{G_I}} + Q_{\mathsf{G_R}} + Q_\mathsf{H}$, with $t_1 \approx t_2 \approx t$ and

$$\varepsilon \leq \varepsilon_1 + \varepsilon_2 + 2S \cdot (\delta_0 + 2\delta_1) + (S + S^2) \cdot 2^{-\lambda} + \mu^2 \cdot 2^{-\gamma_1} \\ + S(S + Q_{\mathsf{G_I}} + Q_{\mathsf{G_R}} + Q_\mathsf{H}) \cdot (2^{-\gamma_0} + 2^{-\alpha_0} + 2^{-\alpha_1+1}) \ .$$

Further, for every adversary $\mathcal{A}$ that breaks the $(t, \varepsilon, \mu, S, T, Q_{COR}, Q_{ST})$-IND-FS-St-security of $\mathsf{AKE}_{\mathsf{st,kem}}$, there exist adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ that break $(t_1, \varepsilon_1, S, S, S, Q_{Ver}, Q_{ST})$-OW-PCVA-C security of KEM_0 and

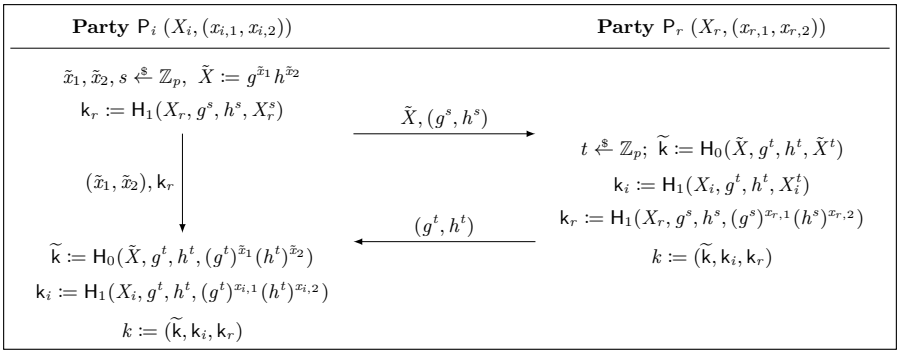

Fig. 11. AKE protocol JKRS. H_0, H_1 are independent random oracles. Protocol $\mathsf{JKRS}_{\mathsf{KC}}$ is obtained by adding the transformation from Figure 9.

$(t_2, \varepsilon_2, \mu, S, S, 2Q_{Ver}, Q_{CoR}, Q_{St})$-OW-PCVA-CR *security of* KEM_1*, where* $Q_{Ver} \leq S + Q_{\mathsf{G}_\mathsf{I}} + Q_{\mathsf{G}_\mathsf{R}} + Q_{\mathsf{H}}$*, with* $t_1 \approx t_2 \approx t$ *and*

$$\begin{aligned}\varepsilon \leq \varepsilon_1 + \varepsilon_2 + 2S \cdot (\delta_0 + 2\delta_1) + (\mu^2 + S^2 + \mu Q_{\mathsf{G}_{\mathsf{stI}}} + 2SQ_{\mathsf{G}_{\mathsf{stI}}}) \cdot 2^{-\kappa} + \mu^2 \cdot 2^{-\gamma_1} \\ + (S + S^2) \cdot 2^{-\lambda} + S(Q_{\mathsf{G}_\mathsf{R}} + Q_{\mathsf{G}_\mathsf{I}} + Q_{\mathsf{H}} + S) \cdot (2^{-\gamma_0} + 2^{-\alpha_0} + 2^{-\alpha_1+1}) \ .\end{aligned}$$

Instantiation with Non-Committing KEM. We can use a non-committing KEM as defined in [JKRS21] to instantiate a verifiable AKE protocol very efficiently, e. g., from DDH (cf. protocol JKRS in Figure 11). We can easily show that a non-committing KEM implies OW-PCVA-CR security of that KEM. In Appendix A, we recall the formal definition of NC-CCA security for KEMs from [JKRS21] and show the implication. Adding key confirmation as described in Figure 9 then yields protocol $\mathsf{JKRS}_{\mathsf{KC}}$.

Security of $\mathsf{JKRS}_{\mathsf{KC}}$. We now establish security of protocol $\mathsf{JKRS}_{\mathsf{KC}}$. Since the JKRS protocol is perfectly correct, so is $\mathsf{JKRS}_{\mathsf{KC}}$. Further, public keys and messages have $\log(p)$ bits entropy. Security is based on the DDH assumption which asks to distinguish between (g^x, g^y, g^{xy}) and (g^x, g^y, g^z) for $x, y, z \xleftarrow{\$} \mathbb{Z}_p$.

Theorem 4. *Let* $\mathsf{JKRS}_{\mathsf{KC}}$ *be defined as in Figure 11, where* $\mathsf{G}_\mathsf{I}, \mathsf{G}_\mathsf{R} : \{0,1\}^* \to \{0,1\}^\lambda$*,* $\mathsf{H} : \{0,1\}^* \to \mathcal{K}$*,* $\mathsf{H}_0 : \{0,1\}^* \to \mathsf{KEM}_0.\mathcal{K}$*,* $\mathsf{H}_1 : \{0,1\}^* \to \mathsf{KEM}_1.\mathcal{K}$*,* $\mathsf{G}_{\mathsf{stI}} : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^{d_\mathsf{I}}$*, and* $\mathsf{G}_{\mathsf{stR}} : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^{d_\mathsf{R}}$ *are modeled as random oracles.*

For every adversary $\mathcal{A}$ *that breaks the* $(t, \varepsilon, \mu, S, T, Q_{CoR}, Q_{St})$-IND-FS-St*-security of* $\mathsf{JKRS}_{\mathsf{KC}}$*, there exists an adversary* $\mathcal{B}$ *that breaks* (t', ε')-DDH *with* $t' \approx t$ *and*

$$\begin{aligned}\varepsilon \leq \varepsilon' + (\mu^2 + S^2 + \mu Q_{\mathsf{G}_{\mathsf{stI}}} + 2SQ_{\mathsf{G}_{\mathsf{stI}}}) \cdot 2^{-\kappa} + \mu^2 \cdot 2^{-\log(p)} \\ + (S + S^2) \cdot 2^{-\lambda} + S(Q_{\mathsf{G}_\mathsf{R}} + Q_{\mathsf{G}_\mathsf{I}} + Q_{\mathsf{H}} + Q_{\mathsf{H}_0} + Q_{\mathsf{H}_1} + S + 1) \cdot 2^{-\log(p)} \ ,\end{aligned}$$

where Q_h *is the number of queries to the respective random oracle* h*.*

The theorem follows from Theorems 2 and 6 and Lemma 2 in combination with [JKRS21, Theorem 5], where the latter deals with the optimization that only one ciphertext is sent in the second round.

Instantiation from Lattices. We can also instantiate the KEM-based verifiable AKE protocol using lattices assumptions. The scheme $\mathsf{KEM}_{\mathsf{LWE}}$ described in [PWZ23a, Section 3] satisfies OW-ChCCA security which implies OW-PCVA-CR security. This gives us an AKE protocol with key confirmation from LWE secure in the random oracle model.

6.2 The CCGJJ Protocol and its Isogeny-based Variant

It is easy to see that the core protocol from Cohn-Gordon et al. (CCGJJ) [CCG+19] is a verifiable AKE protocol, ignoring the session key hash. For completeness, we provide a formal treatment in Appendix F.

Isogeny-based AKE. The isogeny-based AKE protocol which was independently analyzed by de Kock, Gjøsteen and Veroni [dKGV20] and Kawashima et al. [KTAT20] follows the same blueprint as the CCGJJ protocol, relying on the group action structure of CSIDH [CLM+18] rather than prime-order groups. Thus, we also get an AKE protocol with key confirmation from isogenies, based on the same assumptions as the analysis in [dKGV20,KTAT20]. This is particularly interesting because the only group action based and tightly-secure signature scheme supporting adaptive corruptions [PW22] is rather inefficient.

7 KEM-based AKE with Key Confirmation in the QROM

We analyze FS via key confirmation in the quantum random oracle model (QROM). Following [PWZ23b], we use IND-CCA-secure KEMs in the multi-user, multi challenge settings as building blocks. By the key confirmation technique, we lift the result of Pan, Wagner, and Zeng [PWZ23b] to FS in the QROM. The work of Pan, Wagner, and Zeng only achieves weak FS in the QROM. Our result not only preserves the security loss of their protocol, but also achieves FS and allows multiple TEST queries with single challenge bit, while [PWZ23b] allows at most one single TEST query.

We recall the MC-IND-CCA security and MUC-IND-CCA security of KEMs in Appendix A. We use notations introduced in Section 5 to present our protocol $\mathsf{AKE}_{\mathsf{kem}}$. Let KEM_1 and KEM_0 be two KEM schemes and $\mathsf{G}_\mathrm{I}, \mathsf{G}_\mathsf{R} : \{0,1\}^* \to \{0,1\}^\lambda$, and $\mathsf{H} : \{0,1\}^* \to \mathcal{K}$ be hash functions, where λ is the length of key confirmation tags and $\mathcal{K}$ is the key space of $\mathsf{AKE}_{\mathsf{kem}}$. An overview of our AKE construction $\mathsf{AKE}_{\mathsf{kem}}$ is given in Figure 12. $\mathsf{AKE}_{\mathsf{kem}}$ is essentially the KEM-based AKE protocol in [PWZ23b] adding key confirmation, namely, it is obtained from combining Figure 10 and Figure 5.

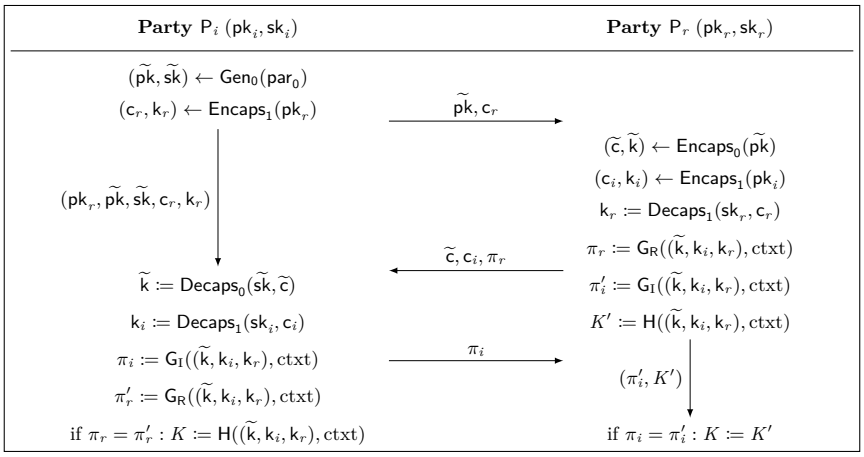

Fig. 12. AKE protocol $\mathsf{AKE}_{\mathsf{kem}}$ from KEM schemes KEM_1, KEM_0, and key confirmation. The context is defined as $\mathrm{ctxt} := (\mathsf{pk}_i, \mathsf{pk}_r, \widetilde{\mathsf{pk}}, \widetilde{\mathsf{c}}, \mathsf{c}_i, \mathsf{c}_r)$. G_I, G_R, and H are independent random oracles.

CORRECTNESS. The correctness of $\mathsf{AKE}_{\mathsf{kem}}$ is due to KEM_1 and KEM_0. Each session of $\mathsf{AKE}_{\mathsf{kem}}$ includes two ciphertexts of KEM_1 and one ciphertext of KEM_0. If KEM_1 is $(1-\delta_1)$-correct and KEM_0 is $(1-\delta_0)$-correct, then by the union bound, $\mathsf{AKE}_{\mathsf{kem}}$ is $(1-2\delta_1-\delta_0)$-correct.

SECURITY. We prove IND-FS security of $\mathsf{AKE}_{\mathsf{kem}}$ based on the MC-IND-CCA security of KEM_1, the MUC-IND-CCA security of KEM_0, and modeling $\mathsf{G}_\mathrm{I}, \mathsf{G}_\mathsf{R}$, and H as quantum-accessible random oracles, as stated in Theorem 5. The proof of Theorem 5 is postponed to Appendix E.

Theorem 5. *Let KEM_0 be $(1-\delta_0)$-correct and have public keys with γ_0 bits of entropy and messages with α_0 bits of entropy. Let KEM_1 be $(1-\delta_1)$-correct and have public keys with γ_1 bits of entropy and messages with α_1 bits of entropy. $\mathcal{K}_0$ and $\mathcal{K}_1$ are the KEM key spaces of KEM_0 and KEM_1, respectively.*

Let $\mathsf{AKE}_{\mathsf{kem}}$ be as defined in Figure 12, where Let $\mathsf{G}_\mathrm{I}, \mathsf{G}_\mathsf{R} : \{0,1\}^* \to \{0,1\}^\lambda$ and $\mathsf{H} : \{0,1\}^* \to \mathcal{K}$. For every adversary $\mathcal{A}$ that breaks the $(t, \varepsilon, \mu, S, T, Q_{COR})$-IND-FS-security of $\mathsf{AKE}_{\mathsf{KC}}$, there exists an adversary $\mathcal{B}_0$ that breaks the $(t'_0, \varepsilon'_0, S, S)$-MUC-IND-CCA security of KEM_0 and an adversary $\mathcal{B}_1$ that breaks the $(t'_1, \varepsilon'_1, S)$-MC-IND-CCA security of KEM_1 with $t'_0 \approx t'_1 \approx t$ and

$$\varepsilon \le 2\varepsilon'_0 + 2\mu\varepsilon'_1 + 2S(\delta_0 + \mu\delta_1) + \mu^2 2^{-\gamma_1} + \mu S 2^{-\lambda+1} + S^2(2^{-\alpha_1} + 2^{-\gamma_0} + 2^{-\alpha_0}) + \frac{2\mu(Q_{\mathsf{G}_\mathsf{R}} + Q_{\mathsf{G}_\mathrm{I}})\sqrt{S}}{\sqrt{|\mathcal{K}_1|}} + \frac{2Q_\mathsf{H}\sqrt{S}}{\sqrt{|\mathcal{K}_0|}},$$

where $Q_{\mathsf{G}_\mathrm{I}}$, $Q_{\mathsf{G}_\mathsf{R}}$ and Q_H are the number of quantum-superposition queries to G_I, G_R and H.

Remark 1 (Implicit Rejection). Following [PWZ23b], when proving Theorem 5, we assume KEM_1 and KEM_0 have implicit rejection [BP18], namely, if the input ciphertext is invalid, then the decapsulation algorithm returns a pseudorandom KEM key. We use implicit-rejection KEM because it simplifies our AKE proof.

To adapt the proof of Theorem 5 to the one that uses explicit-rejection KEMs, we can add extra codes in the games sequence to deal with explicit rejections from KEM. Concretely, upon receiving an invalid KEM ciphertext, the session oracle (e.g., $\textsc{Session}_{\mathsf{R}}$, $\textsc{Der}_{\mathsf{R}}$, or $\textsc{Der}_{\mathsf{I}}$) simply sets the session key as "reject" and returns $\perp$. This can be tightly simulated by MUC-IND-CCA and MC-IND-CCA secure KEMs with explicit rejection, and thus the security bound in Theorem 5 also applies to explicit-rejection KEMs.

Remark 2 (Instantiations with LWE). In [PWZ23b], Pan et al. proposed lattice-based instantiations of MC-IND-CCA-secure KEM and MUC-IND-CCA-secure KEM that have (almost-)tight reduction from the well-known Learning With Errors (LWE) problem in the QROM. Here we only discuss the security loss of these KEM schemes and give the final security loss of our AKE protocol instantiated with these KEM schemes.

Let $\varepsilon_{\mathsf{lwe}}$ be the best computational advantage against LWE assumptions and λ be the security parameter (which decides the security level, the length of message, etc.). The two KEM schemes proposed in [PWZ23b] have asymptotic bounds $\varepsilon'_1 \leq \Theta(\lambda) \cdot \varepsilon_{\mathsf{lwe}}$ and $\varepsilon'_0 \leq \Theta(\lambda) \cdot \varepsilon_{\mathsf{lwe}}$, where ε'_1 and ε'_0 are the computational advantages against MC-IND-CCA security and MUC-IND-CCA security of the KEM schemes in [PWZ23b], respectively. By combining these bounds with the bounds given in Theorem 5, we have

$$\varepsilon \leq \Theta(\lambda) + \Theta(\mu) \cdot \Theta(\lambda) \cdot \varepsilon_{\mathsf{lwe}} = \Theta(\mu) \cdot \Theta(\lambda) \cdot \varepsilon_{\mathsf{lwe}},$$

where ε is the computational advantage against the resulting AKE protocol. This gives us a session-tight and square-root-tight (namely, does not suffer from the square-root security loss) LWE-based instantiation of AKE with full forward secrecy in the QROM.

Acknowledgements. We thank the anonymous reviewers for their valuable comments on better motivating our works and comparisons with the related work. Doreen Riepel was supported in part by Bellare's KACST grant. Jiaxin Pan was supported in part by the Research Council of Norway (RCN) under Project No. 324235, and Runzhi Zeng were supported by the same project from RCN.

References

AFP05. Michel Abdalla, Pierre-Alain Fouque, and David Pointcheval. Password-based authenticated key exchange in the three-party setting. In Serge Vaudenay, editor, *PKC 2005*, volume 3386 of *LNCS*, pages 65–84. Springer, Heidelberg, January 2005.

AHU19. Andris Ambainis, Mike Hamburg, and Dominique Unruh. Quantum security proofs using semi-classical oracles. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 269–295. Springer, Heidelberg, August 2019.

BBM00. Mihir Bellare, Alexandra Boldyreva, and Silvio Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 259–274. Springer, Heidelberg, May 2000.

BDF+11. Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 41–69. Springer, Heidelberg, December 2011.

BP18. Daniel J. Bernstein and Edoardo Persichetti. Towards KEM unification. Cryptology ePrint Archive, Report 2018/526, 2018. https://eprint.iacr.org/2018/526.

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.

BR06. Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, Heidelberg, May / June 2006.

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.

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.

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.

CW13. Jie Chen and Hoeteck Wee. Fully, (almost) tightly secure IBE and dual system groups. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 435–460. Springer, Heidelberg, August 2013.

DFW20. Cyprien Delpech de Saint Guilhem, Marc Fischlin, and Bogdan Warinschi. Authentication in key-exchange: Definitions, relations and composition. In Limin Jia and Ralf Küsters, editors, *CSF 2020 Computer Security Foundations Symposium*, pages 288–303. IEEE Computer Society Press, 2020.

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.

DGJL21. Denis Diemert, Kai Gellert, Tibor Jager, and Lin Lyu. More efficient digital signatures with tight multi-user security. In Juan Garay, editor, *PKC 2021, Part II*, volume 12711 of *LNCS*, pages 1–31. Springer, Heidelberg, May 2021.

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.

dKGV20. 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, *SAC 2020*, volume 12804 of *LNCS*, pages 451–479. Springer, Heidelberg, October 2020.

FGSW16. Marc Fischlin, Felix Günther, Benedikt Schmidt, and Bogdan Warinschi. Key confirmation in key exchange: A formal treatment and implications for TLS 1.3. In *2016 IEEE Symposium on Security and Privacy*, pages 452–469. IEEE Computer Society Press, May 2016.

FHKP13. Eduarda S. V. Freire, Dennis Hofheinz, Eike Kiltz, and Kenneth G. Paterson. Non-interactive key exchange. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 254–271. Springer, Heidelberg, February / March 2013.

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, Heidelberg, August 2023.

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.

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.

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.

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.

JZC+18. Haodong Jiang, Zhenfeng Zhang, Long Chen, Hong Wang, and Zhi Ma. IND-CCA-secure key encapsulation mechanism in the quantum random oracle model, revisited. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 96–125. Springer, Heidelberg, August 2018.

KLS18. Eike Kiltz, Vadim Lyubashevsky, and Christian Schaffner. A concrete treatment of Fiat-Shamir signatures in the quantum random-oracle model. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 552–586. Springer, Heidelberg, April / May 2018.

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.

KTAT20. Tomoki Kawashima, Katsuyuki Takashima, Yusuke Aikawa, and Tsuyoshi Takagi. An efficient authenticated key exchange from random self-reducibility on CSIDH. In Deukjo Hong, editor, *ICISC 20*, volume 12593 of *LNCS*, pages 58–84. Springer, Heidelberg, December 2020.

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, *ASI-*

ACRYPT 2020, Part II, volume 12492 of *LNCS*, pages 785–814. Springer, Heidelberg, December 2020.

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.

PQR22. Jiaxin Pan, Chen Qian, and Magnus Ringerud. Signed (group) diffie-hellman key exchange with tight security. *Journal of Cryptology*, 35(4):26, October 2022.

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, Heidelberg, March 2022.

PWZ23a. Jiaxin Pan, Benedikt Wagner, and Runzhi Zeng. Lattice-based authenticated key exchange with tight security. In *CRYPTO 2023*, LNCS. Springer, Heidelberg, August 2023.

PWZ23b. Jiaxin Pan, Benedikt Wagner, and Runzhi Zeng. Tighter security for generic authenticated key exchange in the qrom. In *ASIACRYPT 2023*, LNCS. Springer, Heidelberg, December 2023. https://eprint.iacr.org/2023/1380.

SXY18. Tsunekazu Saito, Keita Xagawa, and Takashi Yamakawa. Tightly-secure key-encapsulation mechanism in the quantum random oracle model. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 520–551. Springer, Heidelberg, April / May 2018.

Unr14. Dominique Unruh. Revocable quantum timed-release encryption. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 129–146. Springer, Heidelberg, May 2014.

A Definition of KEM

Definition 5 (KEM). *A key encapsulation mechanism (KEM) scheme* KEM *consists of four algorithms* $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ *and a key space* $\mathcal{K}$ *that is assumed to be efficiently recognizable. The algorithms work as follows:*

- *The setup algorithm* Setup*, on input the security parameter* λ*, outputs system parameters* par*.*
- *The key generation algorithm* Gen*, on input the parameter* par*, outputs a public and secret key pair* $(\mathsf{pk}, \mathsf{sk})$*. We assume that* par *is publicly accessible by other algorithms; therefore, for the sake of simplicity, we do not explicitly include* par *in inputs of the following two algorithms.*
- *The encapsulation algorithm* Encaps*, on input* pk*, outputs a ciphertext* c *and a key* $\mathsf{k} \in \mathcal{K}$*.*
- *The decapsulation algorithm* Decaps*, on input* sk *and a ciphertext* c*, outputs a key* $\mathsf{k} \in \mathcal{K}$ *or a rejection symbol* $\perp \notin \mathcal{K}$*.*

Definition 6 (Correctness of KEM). *A KEM scheme* $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ *is* ρ*-correct if*

$$\Pr\left[\mathsf{Decaps}(\mathsf{sk}, \mathsf{c}) = \mathsf{k} \;:\; \begin{array}{l} \mathsf{par} \leftarrow \mathsf{Setup}(1^\lambda), \\ (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{par}), \\ (\mathsf{c}, \mathsf{k}) \leftarrow \mathsf{Encaps}(\mathsf{pk}) \end{array}\right] \geq \rho,$$

where the probability is taken over the randomness of Setup*,* Gen*, and* Encaps*.*

To construct the KEM-based AKE in the classical ROM in Section 6, we define three one-way security of KEM: OW-PCVA, OW-PCVA-C, and OW-PCVA-CR. The latter is a weaker variant of OW-ChCCA security from [PWZ23a] since it does not require the decryption oracle but only has a ciphertext validity oracle.

Definition 7 (OW Security of KEM). *Let* $\mathsf{ATK} \in \{\mathsf{PCVA}, \mathsf{PCVA\text{-}C}, \mathsf{PCVA\text{-}CR}\}$*. We define games* OW-ATK *in Figure 13. Let* μ *be the number of users and* $Q_{\textsc{Enc}}$ *be the number of queries to* Enc*,* $Q_{\textsc{Cv}}$ *be the number of queries to* Cvo*,* $Q_{\textsc{Ch}}$ *be the number of queries to* Check*,* $Q_{\textsc{Cor}}$ *be the number of queries to* Corr*, and* $Q_{\textsc{Rev}}$ *be the number of queries to* Reveal*.*

A KEM *is* $(t, \varepsilon', \mu, Q_{\textsc{Enc}}, Q_{\textsc{Cv}}, Q_{\textsc{Ch}}, Q_{\textsc{Cor}}, Q_{\textsc{Rev}})$*-*OW-ATK*-secure if for all adversaries* $\mathcal{A}$ *attacking the protocol in time* t*, we have*

$$\left|\Pr[\mathsf{OW\text{-}ATK}^{\mathcal{A}}_{\mathsf{KEM}} \Rightarrow 1] \leq \varepsilon' \right. .$$

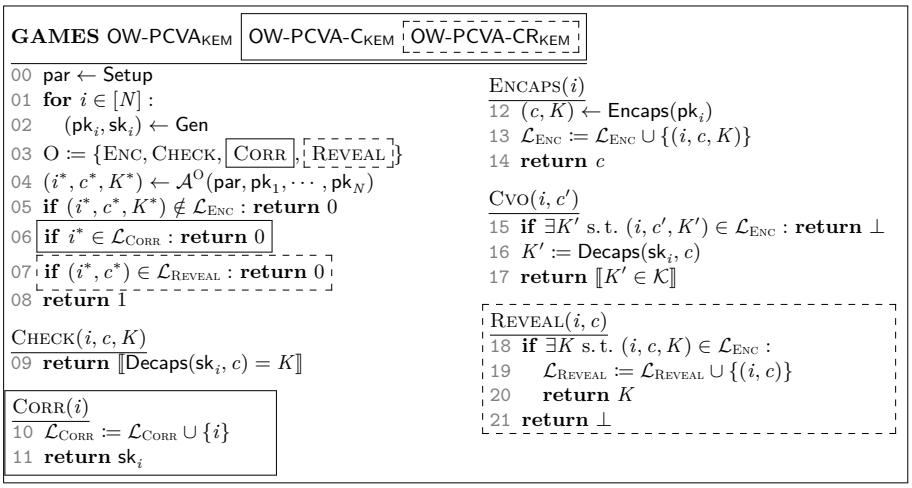

Fig. 13. Games OW-PCVA, OW-PCVA-C (with code in solid boxes) and OW-PCVA-CR (with code in solid and dashed boxes) for KEM.

Remark 3. By definition we have $Q_{\text{COR}} = 0$ if $\mathsf{ATK} = \mathsf{PCVA}$ as well as $Q_{\text{REV}} = 0$ if $\mathsf{ATK} \in \{\mathsf{PCVA}, \mathsf{PCVA\text{-}C}\}$, so we will simply omit the parameters in these cases. It is easy to see that PCVA-CR security implies OW-PCVA-C security which implies OW-PCVA security.

We also define Non-Committing KEMs [JKRS21] and show the relation to OW-PCVA-CR security which we use for protocol $\mathsf{JKRS}_{\mathsf{KC}}$.

Definition 8 (μ-Receiver Non-Committing KEM). *Let* $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ *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 14. The simulator* Sim *is only used in* $\mathsf{NC}_{\mathsf{sim}}$*.*

Let μ be the number of users and Q_{ENC} be the number of queries to Encaps*, Q_{OPEN} be the number of queries to* Open*, Q_{DEC} be the number of queries to* Decaps*, Q_{NCRO} be the number of queries to random oracles.*

We say a KEM *is $(t, \varepsilon', \mu, Q_{\text{ENC}}, Q_{\text{OPEN}}, Q_{\text{DEC}}, Q_{\text{NCRO}})$-*NC-CCA*-secure if for all adversaries $\mathcal{A}$ attacking the protocol in time t, we have*

$$\left|\Pr[\mathsf{NC}^{\mathcal{A}}_{\mathsf{real}}(\lambda) \Rightarrow 1] - \Pr[\mathsf{NC}^{\mathcal{A}}_{\mathsf{sim}}(\lambda) \Rightarrow 1]\right| \leq \varepsilon'$$

GAMES $\mathsf{NC}^{\mathcal{A}}_{\mathsf{real}}(\lambda)$ and $\mathsf{NC}^{\mathcal{A}}_{\mathsf{sim}}(\lambda)$

00 $\mathsf{par} \leftarrow \mathsf{Setup}(1^\lambda)$
01 **for** $i \in [N]$:
02 $\quad(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(\mathsf{par})$
03 $\quad(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{SimGen}(\mathsf{par})$
04 $\quad\mathsf{opened}_i := \mathbf{false}$
05 $\quad\mathcal{D}_i := \varnothing, \mathcal{C}_i := \varnothing, \mathcal{CK}_i := \varnothing, \mathcal{H}_i := \varnothing$
06 $b \leftarrow \mathcal{A}^{\text{O}}(\mathsf{par}, (\mathsf{pk}_i)_{i\in[\mu]})$
07 **return** b

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

Open$(i \in [\mu])$
15 $\mathsf{opened}[i] := \mathbf{true}$
16 **return** sk_i

Encaps$(i \in [\mu])$
17 $(\mathsf{c}, K) \leftarrow \mathsf{Encaps}^{\mathsf{H}_i}(\mathsf{pk}_i)$
18 $\mathsf{c} \leftarrow \mathsf{SimEncaps}(\mathsf{pk}_i, \mathsf{sk}_i)$
19 $K \xleftarrow{\$} \mathcal{K}$
20 $\mathcal{CK}_i := \mathcal{CK}_i \cup \{(\mathsf{c}, K)\}$
21 $\mathcal{C}_i := \mathcal{C}_i \cup \{\mathsf{c}\}$
22 **return** (c, K)

Decaps$(i \in [\mu], \mathsf{c})$
23 **if** $\mathsf{c} \in \mathcal{C}_i$: **return** $\perp$
24 $K := \mathsf{Decaps}^{\mathsf{H}_i}(\mathsf{sk}_i, \mathsf{c})$
25 $\mathcal{D}_i := \mathcal{D}_i \cup \{\mathsf{c}\}$
26 **return** K

Fig. 14. $\mathsf{NC}^{\mathcal{A}}_{\mathsf{real}}(\lambda)$ and $\mathsf{NC}^{\mathcal{A}}_{\mathsf{sim}}(\lambda)$ for an adversary $\mathcal{A}$ and $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ with simulator algorithms $(\mathsf{SimGen}, \mathsf{SimEncaps}, \mathsf{SimHash})$. Each user i is related to a random oracle H_i, and random oracles $\mathsf{H}_1, ..., \mathsf{H}_\mu$ are independent. Algorithms Encaps and Decaps running on user i have oracle access to H_i, but not SimEncaps. $\mathcal{A}$ has access to $\text{O} := \{\mathsf{H}_1, ..., \mathsf{H}_\mu, \text{Encaps}, \text{Decaps}, \text{Open}\}$. Highlighted lines are only executed in $\mathsf{NC}^{\mathcal{A}}_{\mathsf{sim}}(\lambda)$.

Theorem 6. *For every adversary $\mathcal{A}$ that breaks the $(t, \varepsilon, \mu, Q_{\text{ENC}}, Q_{\text{CH}}, Q_{\text{COR}}, Q_{\text{REV}})$-*OW-PCVA-CR *security of* KEM*, there exists an adversary $\mathcal{B}$ that breaks $(t', \varepsilon', \mu, Q_{\text{ENC}}, Q_{\text{CH}}, Q_{\text{COR}})$-*NC-CCA *security of* KEM *with $t' \approx t$ and $\varepsilon \leq \varepsilon'$.*

Proof (Sketch). The proof is straightforward so we only sketch it here. Let $\mathcal{A}$ be an adversary against OW-PCVA-CR. Then we construct an adversary $\mathcal{B}$ aiming to distinguish whether it is in the $\mathsf{NC}_{\mathsf{real}}$ game or in the $\mathsf{NC}_{\mathsf{sim}}$ game as follows. $\mathcal{B}$ gets μ public keys from its challenger and forwards them to $\mathcal{A}$. It also forwards encryption queries to its own encryption oracle, receives (c, k) and gives c to $\mathcal{A}$. Corruption queries can be forwarded. For reveal queries, $\mathcal{B}$ just outputs the key k it received earlier. The non-committing property ensures that this is the correct key in case the user is corrupted. For check queries, $\mathcal{B}$ either asks its decryption oracle (if c is new) or (if c is a challenge) just compares to k it received and again relies on the non-committing property. When $\mathcal{A}$ outputs a candidate k', $\mathcal{B}$ decides it is in the real game if k' equals the key it received from encryption and otherwise it decides it is in the simulated game.

Finally, we define MC-IND-CCA security and MUC-IND-CCA security of KEMs, which will be used to construct the KEM-based AKE in the QROM in Section 7.

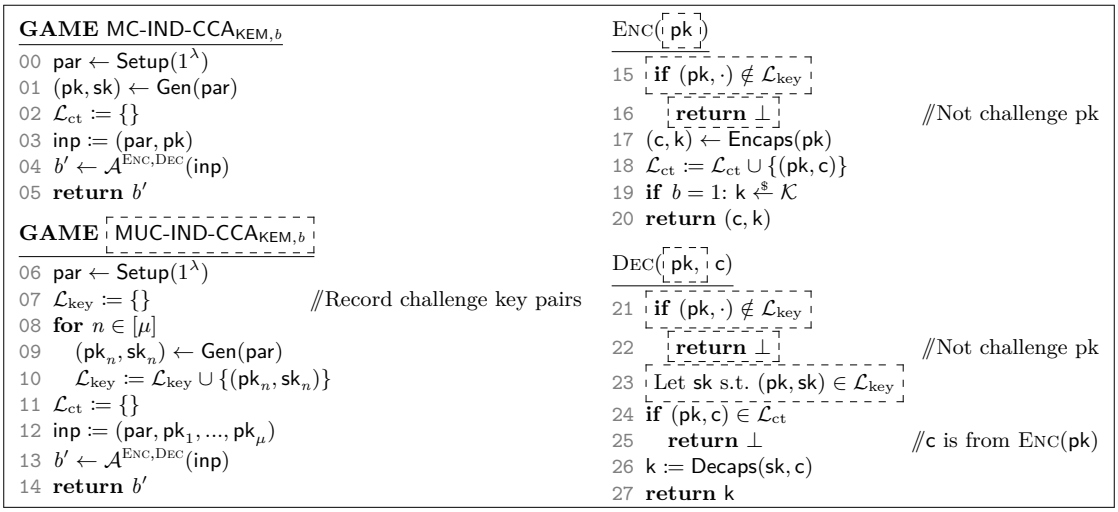

Fig. 15. Games MC-IND-CCA and MUC-IND-CCA. μ is the number of user in the game MUC-IND-CCA. List $\mathcal{L}_{\text{ct}}$ is used to record all challenge ciphertexts generated by Enc. Dashed parts in Enc and Dec are only executed in game MUC-IND-CCA.

Definition 9 (IND-CCA Security of KEM). *We define games* MC-IND-CCA *and* MUC-IND-CCA *in Figure 15. Let S be the number of queries to* Enc *and μ be the number of users in* MUC-IND-CCA.

A KEM *is $(t, \varepsilon', \mu, S)$-*MUC-IND-CCA*-secure if for all adversaries $\mathcal{A}$ attacking the protocol in time t with μ users and S challenge ciphertexts, we have*

$$\left|\Pr[\mathsf{MUC\text{-}IND\text{-}CCA}^{\mathcal{A}}_{\mathsf{KEM},0} \Rightarrow 1] - \Pr[\mathsf{MUC\text{-}IND\text{-}CCA}^{\mathcal{A}}_{\mathsf{KEM},1} \Rightarrow 1]\right| \leq \varepsilon'$$

Similarly, a KEM *is (t, ε', S)-*MC-IND-CCA*-secure if for all adversaries $\mathcal{A}$ attacking the protocol in time t with S challenge ciphertexts, we have*

$$\left|\Pr[\mathsf{MC\text{-}IND\text{-}CCA}^{\mathcal{A}}_{\mathsf{KEM},0} \Rightarrow 1] - \Pr[\mathsf{MC\text{-}IND\text{-}CCA}^{\mathcal{A}}_{\mathsf{KEM},1} \Rightarrow 1]\right| \leq \varepsilon'$$

B Additional Details for the AKE Security Model

We model valid attacks as in [JKRS21]. To cover all possible attacks, we consider four dimensions:

- whether the test session is on the initiator's side (i.e., Type[sID*] ="In") or the responder's side (i.e., Type[sID*] ="Re"),
- all combinations of long-term secret key reveals, taking into account when a corruption happened (cor and peerPreCor variables). For IND-FS-St, we also take state reveals into account.
- the number of matching sessions, i.e. whether the adversary acted passively (matching session) or actively (no matching session).

Table 3 lists all possible attacks from an adversary in game IND-FS. Trivial attacks marked in gray are shown for completeness and *not* considered valid. If the set of variables corresponding to a test session is set as in any row of Table 3, this row will evaluate to **true** in line 12 in Figure 3. We now describe the different attacks in Table 3 in more detail:

Row 1. Here the tested session has a partial matching session, is of type "In", and both parties might be corrupted. Since there is a partial matching session, the adversary has acted passively during the execution of the protocol.[8] Thus, even if both parties were corrupted during the execution, the

[8] Considering partially matching sessions means that we do allow the adversary to drop or modify the last message.

	$\mathcal{A}$ gets (Initiator, Responder)	$\texttt{cor}[i^*]$	$\texttt{cor}[r^*]$	$\texttt{peerPreCor}[\text{sID}^*]$	$\texttt{Type}[\text{sID}^*]$	$\lvert\mathfrak{M}(\text{sID}^*)\rvert$	$\lvert\mathfrak{P}(\text{sID}^*)\rvert$
1.	**(long-term, long-term)**	–	–	–	"In"	–	1
2.	**(long-term, long-term)**	–	–	–	"Re"	1	–
3.	**(long-term, $\perp$)**	–	**T**	**T**	"In"	–	0
4.	**($\perp$, long-term)**	**T**	–	**T**	"Re"	0	–
5.	**(long-term, long-term)**	–	–	**F**	"In"	–	0
6.	**(long-term, long-term)**	–	–	**F**	"Re"	0	–

Table 3. Full table of attacks for adversaries against three-message protocols with IND-FS security. An attack is regarded as an AND conjunction of variables with specified values as shown in the each line, where "–" means that this variable can take arbitrary value and **F** means "false". The trivial attacks where the session's peer is corrupted when the key is derived, and the corresponding variables are set to **T**, are marked with gray. The $\perp$ symbol indicates that the adversary cannot query anything more from this party, as he already possesses the long-term key.

adversary can not break the AKE security without breaking the passive security of the underlying protocol. Hence it should make no difference if the parties were corrupted before or after the key was computed, and the `cor` and `peerPreCor` columns can take any value.

Row 2. This attack is similar to the one above, the only difference is the session type and that a session of type "Re" will have a (fully) matching session.

Row 3. Here, the responder of the session was corrupted when the initiator computed its key, and there is no (partially) matching session. This means that the adversary has performed an active attack and changed or reordered the message being sent. This can lead to a trivial attack, because the adversary can impersonate the responder with the corrupted secret key. By knowing the underlying message, he can compute the same session key as the initiator will compute, and test the initiators session. Whether the adversary corrupts the initiator makes no difference, and hence this column can take any value.

Row 4. Similar to the attack above, with the types switched, and hence the initiator was corrupted by the time the responder computed the key. This leads to a trivial attack in the same way.

Row 5. Here there is no (partially) matching session, but we specify that the responder was not corrupted when the initiator computed its key. The adversary can choose whether or not to corrupt the initiator before the responder computes its key. The key point is that whether he can impersonate the initiator or not, he does not know the internal state of the initiator, and to break security he must either break the underlying key exchange protocol, or impersonate the responder and break the authentication directly. Hence, this column can take any value. After the initiators key is computed, it should not matter whether the responder gets corrupted or not, and hence this column can also take any value.

Row 6. Similar to above, but with the types changed so that the initiator was not corrupted when the responder computed its key.

Since rows (3.) and (4.) are trivial wins for the adversary, we exclude these rows and obtain a simplified version Table 2.

Security with State Reveals. Similar ideas apply to Table 4 which captures valid attacks for adversaries in game IND-FS-St.

In game IND-FS-St, we allow the adversary to reveal session states. Table 4 listed all attacks allowed in game IND-FS-St. All these attacks capture FS, KCI, and state-reveal (ST) attacks. Similar to Table 2, Table 4 is obtained from considering all possible attacks and then reducing all trivial attacks. In the state-reveal model, if the adversary obtain both the session state and the long-term key of session owner, then the session is trivially broken since there is no any secret information in the session.

	$\mathcal{A}$ gets (Initiator, Responder)	$\mathtt{cor}[i^*]$	$\mathtt{cor}[r^*]$	$\mathtt{peerPreCor}[\mathrm{sID}^*]$	$\mathtt{Type}[\mathrm{sID}^*]$	$\lvert\mathfrak{M}(\mathrm{sID}^*)\rvert$	$\mathtt{revST}[\mathrm{sID}^*]$	$\exists \mathrm{sID}' \in \mathfrak{M}(\mathrm{sID}^*) : \mathtt{revST}[\mathrm{sID}']$	$\lvert\mathfrak{P}(\mathrm{sID}^*)\rvert$	$\exists \mathrm{sID}' \in \mathfrak{P}(\mathrm{sID}^*) : \mathtt{revST}[\mathrm{sID}']$
1.	**(long-term, long-term)**	–	–	–	"In"	–	**F**	**F**	1	**F**
2.	**(long-term, long-term)**	–	–	–	"Re"	1	**F**	**F**	–	**F**
5.	**(long-term, long-term)**	–	–	**F**	"In"	–	**F**	n/a	0	n/a
6.	**(long-term, long-term)**	–	–	**F**	"Re"	0	**F**	n/a	–	n/a
7.	**(long-term, state)**	–	**F**	**F**	"In"	–	**F**	–	–	–
8.	**(long-term, state)**	–	**F**	–	"Re"	1	–	**F**	–	**F**
9.	**(long-term, state)**	–	**F**	**F**	"Re"	0	–	**F**	–	**F**
10.	**(state, long-term)**	**F**	–	–	"In"	–	–	**F**	1	**F**
11.	**(state, long-term)**	**F**	–	**F**	"In"	–	–	**F**	0	**F**
12.	**(state, long-term)**	**F**	–	**F**	"Re"	–	**F**	–	–	–
13.	**(state, state)**	**F**	**F**	**F**	"In"	–	–	–	–	–
14.	**(state, state)**	**F**	**F**	**F**	"Re"	–	–	–	–	–

Table 4. Table of attacks for adversaries against three-message protocols in $\mathsf{IND\text{-}FS\text{-}St}$. This table is optimized, namely, it is obtained from considering all possible attacks and then removing redundant row. An attack is regarded as an AND conjunction of variables with specified values as shown in the each line, where "–" means that this variable can take arbitrary value, **F** means "false", and "n/a" indicates that there is no state which can be revealed as no (partial) matching session exists.

C Proof of Theorem 2

Proof (Theorem 2). Let $\mathcal{A}$ be an adversary against $\mathsf{IND\text{-}FS\text{-}St}$ security of $\mathsf{AKE}_{\mathsf{stKC}}$. We use the sequence of games G_0-G_4 to finish the proof.

GAME G_0. This game is the original $\mathsf{AKE}_{\mathsf{stKC}}$ game, however we exclude collisions of public keys, messages, and state encryption keys. Similar to the game G_0 in the proof of Theorem 1, we have

$$\Pr[\mathsf{G}_0^{\mathcal{A}} \Rightarrow 1] \leq \Pr[\mathsf{IND\text{-}FS\text{-}St}^{\mathcal{A}}_{\mathsf{AKE}_{\mathsf{stKC}}} \Rightarrow 1] + \mu^2(2^{-\gamma} + 2^{-\kappa}) + S^2(2^{-\alpha} + 2^{-\lambda} + 2^{-\kappa}) \ .$$

This also means that, in G_0, there can be at most one (partially) matching session for each session, and every session sID has a unique state encryption one-time key IV corresponding.

GAME G_1. We postpone the computation of one-time pads used in encrypting session states (i.e., $\mathsf{G}_{\mathsf{stI}}(\mathsf{IV}, \mathsf{s})$). We also ensure that π_r, π_i, and K have not been queried to the respective random oracle before they are determined (as we did in the G_1 in the proof of Theorem 5).

Concretely, we generate the encrypted states φ_i by independently and uniformly sampling (cf. line 51). We use a list $\mathtt{ST}'$ to store the state information that will be used to simulate $\textsc{Der}_{\mathrm{I}}$ (cf. line 61) and patch $\mathsf{G}_{\mathsf{stI}}$. In $\mathtt{ST}'$, each entry $\mathtt{ST}'[i, \mathsf{IV}]$ stores the session sID corresponding to IV, the encrypted state φ_i, and the unencrypted state st_i (cf. line 52). If $\mathcal{A}$ has found the one-time pad $\mathsf{G}_{\mathsf{stI}}(\mathsf{IV}_i, \mathsf{s}_i)$ used to encrypt st_i (cf. lines 141 to 143), then we recover st_i from the list $\mathtt{ST}'$ and patch $\mathsf{G}_{\mathsf{stI}}(\mathsf{IV}_i, \mathsf{s}_i) \coloneqq \varphi_i \oplus \mathrm{st}_i$.

Similar to the game G_1 in the proof of Theorem 1, since AKE' has α bits of entropy of protocol messages and IV_i is sampled uniformly at random, we have

$$|\Pr[\mathsf{G}_0^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathsf{G}_1^{\mathcal{A}} \Rightarrow 1]| \leq (Q_{\mathsf{G}_{\mathsf{R}}} + Q_{\mathsf{G}_{\mathrm{I}}} + Q_{\mathsf{H}}) \cdot S \cdot 2^{-\alpha} + Q_{\mathsf{G}_{\mathsf{stI}}} \cdot S \cdot 2^{-\kappa}$$

GAME G_2. We use a list $\mathtt{queryOTP}$ to record whether $\mathcal{A}$ has found the one-time pad for encrypting state (cf. line 144), and if $\mathcal{A}$ found the one-time pad of session sID while the owner user of sID is uncorrupted or the state of sID is unrevealed, then we raise a flag $\mathsf{QuerySt}$ and abort the game (cf. lines 41 to 42).

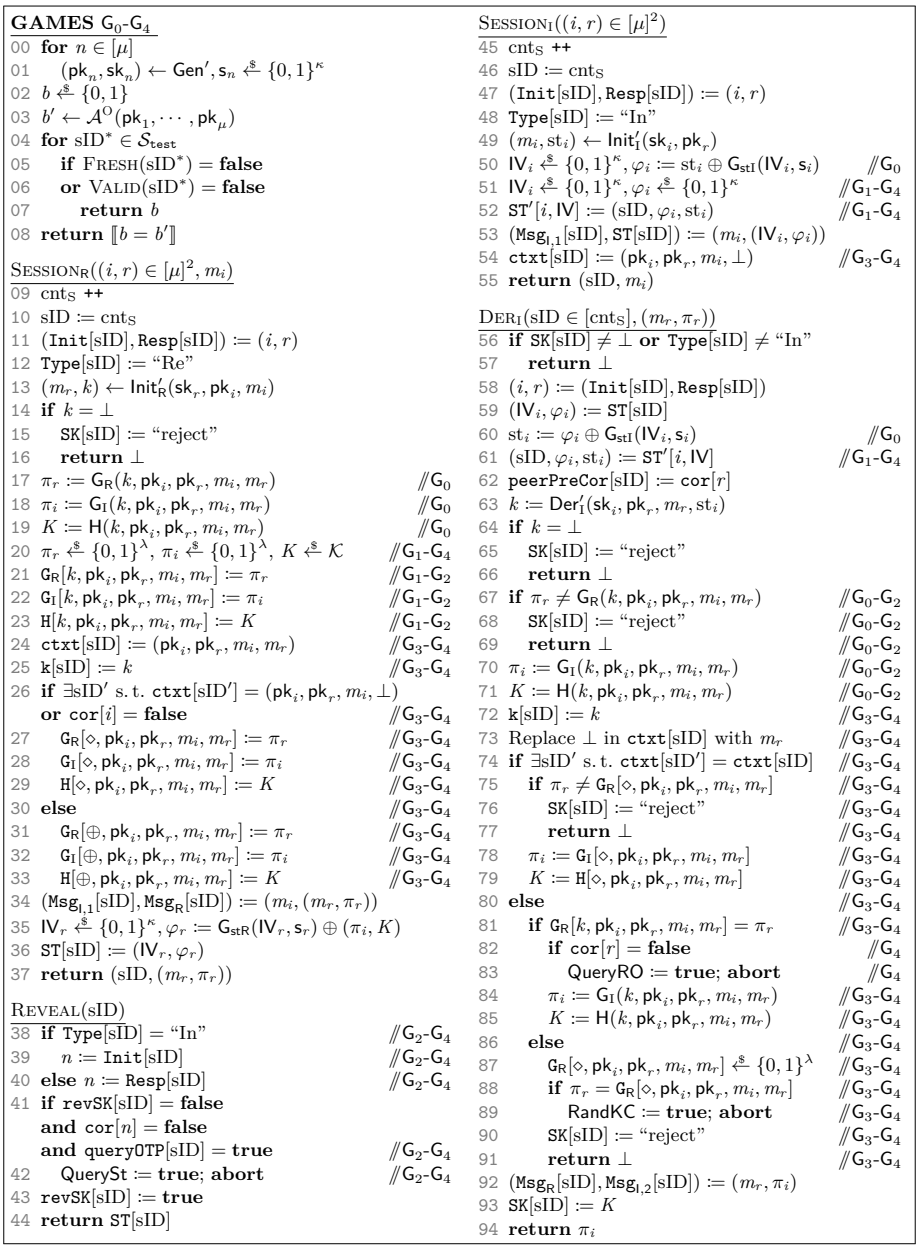

Fig. 16. Games G_0-G_4 for the proof of Theorem 2. $\mathcal{A}$ has access to oracles $\mathrm{O} \coloneqq \{\textsc{SessionI}, \textsc{SessionR}, \textsc{DerI}, \textsc{DerR}, \textsc{Reveal}, \textsc{Corr}, \textsc{Test}, \mathsf{G}_\mathrm{I}, \mathsf{G}_\mathsf{R}, \mathsf{H}, \mathsf{G}_{\mathsf{st}\mathrm{I}}, \mathsf{G}_{\mathsf{st}\mathsf{R}}\}$. Helper procedures Fresh and Valid are defined in Figure 3. $\mathsf{G}_{\mathsf{st}\mathsf{R}}$ is defined as a usual RO simulation.

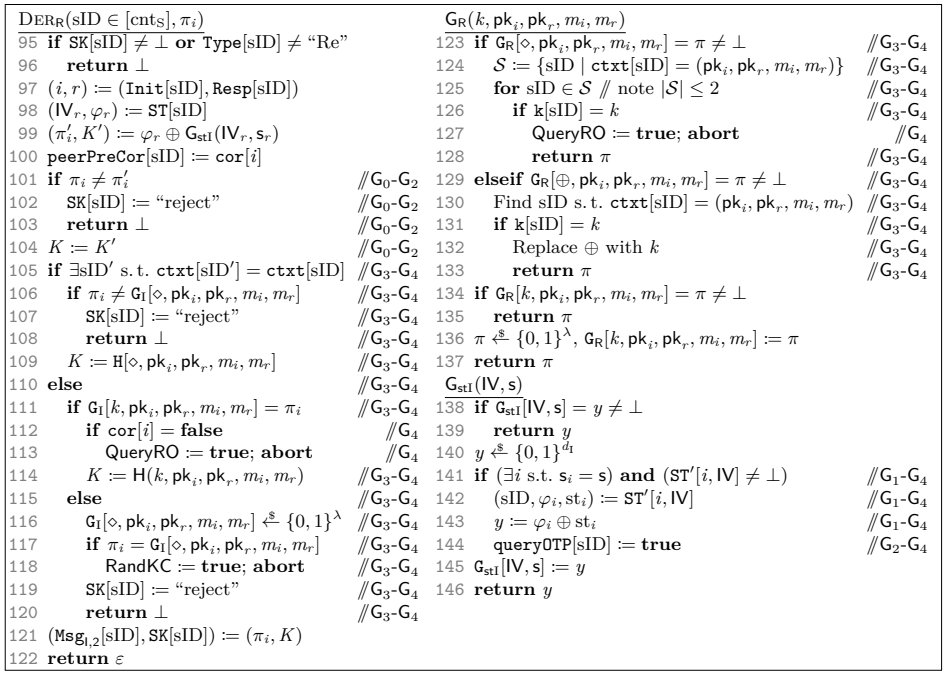

Fig. 17. Oracles for games G_0-G_4 for the proof of Theorem 1. G_{I} and H are defined analogously to G_{R}.

To trigger $\mathsf{QuerySt}$, $\mathcal{A}$ has to queries $\mathsf{G}_{\mathsf{stI}}$ on $(\mathsf{IV}_i, \mathsf{s}_i)$ (for some user i) while the session state of sID (the session that corresponds to IV_i) is unrevealed or the user i is uncorrupted. Since both IV_i and s_i are generated independently and uniformly at random, if user i is uncorrupted (resp., sID is unrevealed), then s_i (resp., IV_i) is uniformly random in $\mathcal{A}$'s view. Therefore, by a union bound, we have

$$|\Pr[\mathsf{G}_1^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathsf{G}_2^{\mathcal{A}} \Rightarrow 1]| \leq \Pr[\mathsf{QuerySt}] \leq (\mu + S) \cdot Q_{\mathsf{G}_{\mathsf{stI}}} \cdot 2^{-\kappa}$$

By introducing this abort event, now we can assure that, unless $\mathcal{A}$ both corrupts the owner user and reveals the session state, it cannot learn the unencrypted state generated by AKE'. This fact is crucial for reduction to the $\mathsf{OW\text{-}VwFS\text{-}St}$ security of AKE', since it forces that a fresh and valid session of $\mathsf{AKE}_{\mathsf{stKC}}$ will correspond a fresh and valid session of AKE'.

Another important fact implied by this game is, unless $\mathcal{A}$ both corrupts the owner user and reveals the session state, it cannot learn any useful information from state-reveal oracle, because the outputs of the state-reveal oracle in this case are all uniformly random. This makes the remaining security proof of $\mathsf{AKE}_{\mathsf{stKC}}$ similar to the one of $\mathsf{AKE}_{\mathsf{KC}}$.

Game G_3. This game is similar to the game G_2 in the proof of Theorem 1, except that here we need to handle states reveal. To prepare for the reduction to $\mathsf{OW\text{-}VwFS\text{-}St}$, we use the proof strategy of G_2 in Theorem 1 to compute π_r, π_i and K without using k explicitly, and raise a flag RandKC if the adversary successfully forges a valid key confirmation tag without the random oracle being queried. This change forces that $\mathcal{A}$ cannot compute valid key confirmation tags of valid sessions without querying random oracle.

We claim that state-reveal oracle does not help $\mathcal{A}$ to trigger RandKC. This is because both initiator session state and responder session state do not include any information to help $\mathcal{A}$ to forge a valid key confirmation tag without querying random oracle. Therefore, by the same arguments used in G_2 in Theorem 1, we have

$$|\Pr[\mathsf{G}_2^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathsf{G}_3^{\mathcal{A}} \Rightarrow 1]| \leq \Pr[\mathsf{RandKC}] \leq S \cdot 2^{-\lambda} + S \cdot \delta \ .$$

Game G_4. This game is similar to the game G_3 in the proof of Theorem 1, except that here we need to handle states reveal. We raise a flag $\mathsf{QueryRO}$ if the adversary ever queries the random oracle on a key k of a fresh session.

We first claim that $\mathcal{A}$ does not have any advantage in this game. To see this, we consider all types of valid sessions listed in Table 4.:

– Types 1,2,5,6: These types of attacks do not use the state-reveal oracle, by the argument used in G_3 of Theorem 1, $\mathcal{A}$ dose not have any advantage over test sessions of these types.
– Types 7,12: For type-7 (resp., type-12) test sessions, the peer users are uncorrupted. If such session does not have a partially matching (resp., matching) session, then by line 83 (resp., line 113), $\mathcal{A}$ will triggers QueryRO if it send valid key confirmation tags to such session. That is, unless $\mathcal{A}$ triggers QueryRO, $\mathcal{A}$ cannot complete such session and thus cannot test it.

  If such session has a matching session, then unless $\mathcal{A}$ queried the correct k of such session (which will trigger QueryRO in line 127), $\mathcal{A}$ cannot distinguish whether the returned session key from test oracle is real or random, since it is output of random oracle.
– Types 8,10: In these cases, the session has a (partially) matching session. Similar to types 7 and 12 (with partially matching or matching sessions), $\mathcal{A}$ cannot distinguish whether the returned session key from test oracle is real or random unless it queried the correct key k of such session (and triggers QueryRO).
– Types 9,11: $\mathcal{A}$ cannot complete such session unless it triggers QueryRO. For example, let us consider a type-9 session sID. Since sID does not have matching session and the peer of sID is uncorrupted when $\mathcal{A}$ tampers sID, $\mathcal{A}$ completes this session means that it sent a valid key confirmation tag to sID, which will trigger QueryRO by line 113. Type 11 is symmetric to type 9, and thus a similar argument applies to type-9 sessions.
– Types 13,14: Let sID be a type-13 session sID. By definition, both the owner user and peer user of sID are uncorrupted. If sID has a partially matching session, then $\mathcal{A}$ has to queried the correct key k of sID to distinguish. If sID does not have partially matching session, then $\mathcal{A}$ has to send the valid key confirmation tag to sID to complete the session. Both cases will raise QueryRO. A similar argument applies to type-14 sessions.

Therefore, for any valid session captured in Table 4, unless $\mathcal{A}$ triggers QueryRO, it does not have any advantage to distinguish the outputs from the Test oracle. So, we have

$$\Pr[\mathsf{G}_4^{\mathcal{A}} \Rightarrow 1] = 1/2.$$

Bounding event QueryRO. Now it remains to bound $\Pr[\mathsf{QueryRO}]$. We describe an adversary $\mathcal{B}_{\mathsf{st}}$ against OW-VwFS-St security of the underlying AKE' to bound event QueryRO. $\mathcal{B}_{\mathsf{st}}$ simulates G_4 for $\mathcal{A}$ and if $\mathcal{A}$ triggers QueryRO, then $\mathcal{B}_{\mathsf{st}}$ outputs a solution of its OW-VwFS-St challenge.

A pseudocode description is given in Figure 18. $\mathcal{B}_{\mathsf{st}}$ has a very similar structure with the adversary $\mathcal{B}$ in Figure 8 except for the parts related to session states. For simplicity, here we only describe how $\mathcal{B}_{\mathsf{st}}$ simulate session states and the state-reveal oracle.

$\mathcal{B}_{\mathsf{st}}$ gets as input μ public keys and forwards them to $\mathcal{A}$. To use the state-encryption technique, $\mathcal{B}_{\mathsf{st}}$ also generates random key s_n for each user n. Such keys are independent of $\mathcal{B}_{\mathsf{st}}$'s challenge public keys.

$\mathcal{B}_{\mathsf{st}}$ can simulate queries to oracle $\textsc{Session}_{\mathrm{I}}$ in a straightforward way by querying its own oracle $\textsc{Session}'_{\mathrm{I}}$ which returns (sID, m_i). Since in the OW-VwFS-St game, $\mathcal{B}_{\mathsf{st}}$ does not have the unencrypted session state of AKE', it simply leaves it as unknown (cf. line 61). Other parts are the same as in G_4. When $\mathcal{A}$ queries $\textsc{Der}_{\mathrm{I}}$, $\mathcal{B}_{\mathsf{st}}$ forward the query to $\textsc{Der}'_{\mathrm{I}}$. Although $\mathcal{B}_{\mathsf{st}}$ does have the session state of AKE', it can still use $\textsc{Der}'_{\mathrm{I}}$ to finish the session. It needs to know such AKE' state only if $\mathcal{A}$ found the one-time pad for encrypting this state (which will be described later). $\mathcal{B}_{\mathsf{st}}$ uses $\mathcal{B}$'s technique (in Figure 8) to check whether $\mathcal{A}$ found a one-way solution of $\mathcal{B}_{\mathsf{st}}$'s OW-VwFS-St challenge and simulate the session key and key confirmation tag. Oracle $\textsc{Session}_{\mathsf{R}}$ and $\textsc{Der}_{\mathsf{R}}$ are simulated similarly, looking at G_{I} instead of G_{R}.

Queries to Test will always return the real session key, which is a perfect simulation since session keys are perfectly hidden unless $\mathcal{A}$ triggers QueryRO. Oracle Corr can be simulated in a straightforward way by forwarding the query to $\textsc{Corr}'$. The simulation of G_{R}. G_{I} and H are also similar to the oracles of $\mathcal{B}$ in Figure 8.

It remains to describe the simulation of Rev-State and $\mathsf{G}_{\mathsf{stI}}$:
– $\mathcal{B}_{\mathsf{st}}$ does not have the session states of AKE', so it uses the $\textsc{Rev-State}'$ oracle provided by the OW-VwFS-St game to respond the unencrypted session state. Concretely, if $\mathcal{A}$ both reveal a session state and corrupt the owner and use this information to query $\mathsf{G}_{\mathsf{stI}}$, then $\mathcal{B}_{\mathsf{st}}$ also reveal the state of corresponding session of AKE' (cf. line 115) and patch $\mathsf{G}_{\mathsf{stI}}$ to make the simulation consistent.
– Rev-State is simulated as in G_4. The abort event in Rev-State assures that $\mathcal{B}_{\mathsf{st}}$ will not query $\textsc{Rev-State}'$ unless $\mathcal{A}$ corrupted the owner user and revealed session state. If $\mathcal{A}$ did this, then the

Fig. 18. Adversary $\mathcal{B}_{\mathsf{st}}$ against OW-VwFS-St. $\mathcal{A}$ has access to oracles $\mathrm{O} \coloneqq \{\text{Session}_{\mathsf{I}}, \text{Session}_{\mathsf{R}}, \text{Der}_{\mathsf{I}}, \text{Der}_{\mathsf{R}}, \text{Reveal}, \text{Corr}, \text{Test}, \mathsf{G}_{\mathsf{I}}, \mathsf{G}_{\mathsf{R}}, \mathsf{H}, \mathsf{G}_{\mathsf{stI}}, \mathsf{G}_{\mathsf{stR}}\}$. Procedures Fresh and Valid are defined in Figure 3. Oracles G_{I} and H are defined analogously to G_{R}. $\mathsf{G}_{\mathsf{stR}}$ is defined as a usual RO simulation.

session is invalid. This abort event makes sure that all valid sessions during the reduction will always correspond to a valid session of AKE' in the game OW-VwFS-St.

This concludes the description of adversary $\mathcal{B}_{\mathsf{st}}$. If QueryRO happens during $\mathcal{B}_{\mathsf{st}}$'s simulation, i. e., there exists a random oracle query for a fresh and valid session and correct key k of AKE', then $\mathcal{B}_{\mathsf{st}}$ wins the OW-VwFS-St game. We get

$$\Pr[\mathsf{QueryRO}] \leq \varepsilon' + S\delta \ .$$

Further, $\mathcal{B}_{\mathsf{st}}$ issues at most S queries to Rev-State$'$ (since there are at most S sessions), and at most $(S + Q_{\mathsf{G_I}} + Q_{\mathsf{G_R}} + Q_{\mathsf{H}})$ to KVer. The number of queries to all other oracles is preserved. This completes the proof of Theorem 2.

D Omitted Proofs from Section 6

D.1 Proof of Lemmas 1 and 2

Since the proofs are almost identical, we proof them together, only highlighting the additional changes for Lemma 2.

Proof. For the proof of Lemma 1 let $\mathcal{A}$ be an adversary that breaks the $(t, \varepsilon, \mu, S, T, Q_{\mathrm{Cor}}, Q_{Ver})$-OW-VwFS-security of $\mathsf{AKE}'_{\mathsf{kem}}$. For that of Lemma 2 let $\mathcal{A}$ be an adversary that breaks the $(t, \varepsilon, \mu, S, T, Q_{\mathrm{Cor}}, Q_{Ver}, Q_{\mathrm{St}})$-OW-VwFS-St-security of $\mathsf{AKE}'_{\mathsf{kem}}$. That is $\mathcal{A}$ computes the key $k^* = (\widetilde{\mathsf{k}}^*, \mathsf{k}_i^*, \mathsf{k}_r^*)$ for a fresh and valid sID* according to the game definition.

For the proof we will make a case distinction depending on whether sID* has a matching session and which parties are corrupted. Recall that we have to consider the following three cases for valid attacks:
(1) there exists a matching session, or
(2) there does not exist a matching session, the session is of type "In" and the responder is not corrupted.
(3) there does not exist a matching session, the session is of type "Re" and the initiator is not corrupted.
We will handle case (2) and (3) in one step. Thus, we only have to analyze

$$\begin{aligned}\epsilon \leq{}& \Pr[\mathsf{OW\text{-}VwFS}^{\mathcal{A}}_{\mathsf{AKE}'_{\mathsf{kem}}} \Rightarrow 1 \wedge \text{Case (1)}],\\ &+ \Pr[\mathsf{OW\text{-}VwFS}^{\mathcal{A}}_{\mathsf{AKE}'_{\mathsf{kem}}} \Rightarrow 1 \wedge (\text{Case (2) or Case (3)})]\end{aligned}$$

and analogously for OW-VwFS-St which defines the same valid attacks, except that we also require that the state has not been revealed.

Case (1). In order to analyze this case, we will construct an adversary against OW-PCVA (Lemma 1) and against OW-PCVA-C (Lemma 2) of KEM_0 with S users as given in Figure 19. It simulates all parts related to the long-term keys itself and uses its input and oracles to simulate the ephemeral public keys and ciphertexts. For this, it embeds $\widetilde{\mathsf{pk}}_i$ in the i-th session and whenever this public key is used in a responder session that (partially) matches an initiator session, $\mathcal{B}_1$ queries its challenge oracle Enc to obtain a ciphertext $\widetilde{\mathsf{c}}$. Responder sessions that do not have a partially matching session can be simulated trivially. In order to simulate initiator sessions that do not have a matching session, $\mathcal{B}_1$ queries Cvo in order to determine whether the ciphertext is valid. If this is the case, it stores a placeholder $\diamond$ for k_1.

For Lemma 2, $\mathcal{B}_1$ needs to simulate the Rev-State$'$ oracle. Note that only initiator sessions have a state and that the only unknown value is the corresponding $\widetilde{\mathsf{sk}}_i$ for $\widetilde{\mathsf{pk}}_i$ used in the i-th session. To obtain $\widetilde{\mathsf{sk}}_i$, $\mathcal{B}_1$ simply queries its Corr oracle on i.

Finally, we need to simulate queries to KVer. We use the same helper procedure FindMatch as in the proof of Lemma 6 in order to decide whether the session has a (partially) matching session and to find the corresponding ephemeral public key and ciphertext. If it exists, $\mathcal{B}_1$ queries Check. If the output is **true**, it has found a solution to the OW-PCVA resp. OW-PCVA-C game. For the latter, we also require that the state has not been revealed. If the output is **false** (or the state was revealed), it simply returns the output of Check. If there does not exist a matching session, then $\mathcal{B}_1$ needs to simulate the initiator session correctly. It does so by querying Check on the ciphertext the adversary used in that session. Thus, $\mathcal{B}_1$ queries Check at most Q_{Ver} times. We have $\Pr[\mathsf{OW\text{-}VwFS}^{\mathcal{A}}_{\mathsf{AKE}'_{\mathsf{kem}}} \Rightarrow 1 \wedge \text{Case (1)}] \leq \varepsilon_{\mathcal{B}_1}$ and analogously for OW-VwFS-St.

Fig. 19. Adversary $\mathcal{B}_1$ against OW-PCVA resp. $\boxed{\text{OW-PCVA-C}}$ (including boxes) for the proof of Lemma 1 resp. Lemma 2. Highlight lines show how we embed the challenge from OW-PCVA resp. OW-PCA-C.

CASE (2) AND CASE (3). In order to analyze the case where there is no matching session, we will rely on OW-PCVA-C (Lemma 1) resp. OW-PCVA-CR (Lemma 2) security of the long-term KEM KEM_1. We construct an adversary $\mathcal{B}_2$ in Figure 20. The public keys that $\mathcal{B}_2$ receives will be the long-term public keys. Then in each session $\mathcal{B}_2$ queries its challenge oracle ENC to receive ciphertext c_r (in initiator sessions with responder r) or c_i (in responder sessions with initiator r). It queries CVO to check validity of the ciphertext whenever the ciphertext is new, i. e., it was not previously output by $\mathcal{B}_2$ in any session. Then it sets the session key to $(\widetilde{\mathsf{k}}, \diamond, \diamond)$. Queries to CORR' are answered using the corrupt oracle CORR from the OW-PCVA-C resp. OW-PCVA-CR game.

As in the previous case, we explain how $\mathcal{B}_2$ simulates the $\text{REV-STATE}'$ oracle for Lemma 2. Here, the only unknown value is the key k_r for the ciphertext c_r used in an initiator session. To obtain k_r, $\mathcal{B}_2$ simply queries its REVEAL oracle on (r, c_r).

Queries to KVER are simulated as follows. Since $\mathcal{B}_2$ has not computed k_i and k_r, it needs to check whether they are correct. Note that $\mathcal{A}$ may trivially compute them, so either $\mathcal{B}_2$ just simulates the oracle's output or it stops to solve its own game. For this, $\mathcal{B}_2$ first queries CHECK for both ciphertexts c_i and c_r. We denote the boolean outputs by b_i and b_r. If the session is an initiator session and b_r is **true** and the responder is not corrupted (this corresponds to case (2)), then $\mathcal{B}_2$ stops with $(r, \mathsf{c}_r, \mathsf{k}_r)$ which is a valid solution in the OW-PCVA-C game. For Lemma 2, we additionally require that the state is not revealed and then the solution is valid in the OW-PCVA-CR game. $\mathcal{B}_2$ proceeds similar for responder sessions with

Fig. 20. Adversary $\mathcal{B}_2$ against OW-PCVA-C resp. OW-PCVA-CR (including boxes) for the proof of Lemma 1 resp. Lemma 2. Highlight lines show how we embed the challenge from OW-PCVA-C resp. OW-ChCCA.

initiator i (which corresponds to case (3)). If it has not stopped, then it simply returns **true** if both b_i and b_r are **true**. Otherwise, it returns **false**. Note that the simulation is perfect and that $\mathcal{B}_2$ issues at most $2Q_{Ver}$ queries to Check. We have $\Pr[\mathsf{OW\text{-}VwFS}^{\mathcal{A}}_{\mathsf{AKE}'_{\mathsf{kem}}} \Rightarrow 1 \wedge (\text{Case (2) or Case (3)})] \leq \varepsilon_{\mathcal{B}_2}$ and analogously for OW-VwFS-St, which concludes the proofs of Lemmas 1 and 2.

E QROM Proof of Theorem 5

We first recall the notion of quantum random oracle model (QROM) and some lemmas of interest.

E.1 Quantum Random Oracle Model

In the quantum random oracle model (QROM), hash functions are modelled as publicly quantum-accessible random oracles (see [BDF+11] for more details). Adversaries in the QROM can query ROs on quantum superposition. Let qRO be a random oracle, we denote $\mathcal{A}$ with quantum access to qRO by $\mathcal{A}^{|\mathsf{qRO}\rangle}$.

Lemma 3 ([AHU19] and [PWZ23b, Corollary 2.2]) gives a probabilistic bound for an adversary $\mathcal{A}$ (at most q queries to $|\mathsf{qRO}\rangle$) to distinguish whether it is interacting with random oracle qRO_0 or interacting

with random oracle qRO_1, where $\mathsf{qRO}_0 \backslash \mathcal{S} = \mathsf{qRO}_1 \backslash \mathcal{S}$ (i.e., $\forall x \notin \mathcal{S}$, $\mathsf{qRO}_0(x) = \mathsf{qRO}_1(x)$) and $\mathcal{S}$ is an independently and uniformly random set.

Lemma 3. *Let $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{S} \subseteq \mathcal{X}$ be sets. Let $\mathsf{qRO}_0, \mathsf{qRO}_1 : \mathcal{X} \to \mathcal{Y}$ be random functions satisfying $\forall x \notin \mathcal{S}$, $\mathsf{qRO}_0(x) = \mathsf{qRO}_1(x)$. Let inp be some bitstring. $(\mathcal{S}, \mathsf{qRO}_0, \mathsf{qRO}_1, \mathsf{inp})$ may have arbitrary joint distribution. Let $\mathcal{A}$ be an adversary issuing at most Q_{qro} quantum-superposition queries to random oracle and, on input inp, it outputs either 0 or 1.*

If $\mathcal{S}$ is chosen independently and uniformly at random, then for any PPT adversary $\mathcal{A}$, we have

$$\left|\Pr[1 \leftarrow \mathcal{A}^{|\mathsf{qRO}_0\rangle}(\mathsf{inp})] - \Pr[1 \leftarrow \mathcal{A}^{|\mathsf{qRO}_1\rangle}(\mathsf{inp})]\right| \leq 2Q_{\mathsf{qro}}\sqrt{|\mathcal{S}|/|\mathcal{X}|}$$

Remark 4 (Simulation of QROs). When constructing reduction in the QROM, we need to simulate ROs such that the adversary can issue quantum-superposition queries. Unfortunately, unlike classical ROM, efficient reduction algorithm in the QROM cannot use lazy sampling to simulate quantum random oracles (QROs). Following [JZC+18,KLS18,SXY18,PWZ23b], we do not specify how to simulate QROs. Instead, we assume that reductions have access to some internal quantum random oracles, which can be instantiated by quantum-secure pseudo-random functions or real-world hash functions [KLS18,SXY18].

E.2 Proof of Theorem 5

Proof (Theorem 5). We use games sequence G_0-G_4 (shown in Figure 21) to prove Theorem 5. We only present the codes of modified parts. The full codes of G_0 can be found in Figure 24.

GAME G_0. This game is the same as $\mathsf{IND\text{-}FS}_{\mathsf{AKE}_{\mathsf{kem}}}$, except that we exclude collisions of long-term key pairs $(\mathsf{pk}_i, \mathsf{sk}_i)$, ciphertexts c from KEM_1, ephemeral key pairs $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}})$ and ciphertexts $\widetilde{\mathsf{c}}$ from KEM_0, session keys, and key confirmation tags generated by the game. 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.

$$\begin{aligned}&\left|\Pr[\mathsf{IND\text{-}FS}^{\mathcal{A}}_{\mathsf{AKE}_{\mathsf{kem}}} \Rightarrow 1] - \Pr[\mathsf{G}_0^{\mathcal{A}} \Rightarrow 1]\right| \\ &\qquad \leq \mu^2 2^{\gamma_1} + S^2(2^{-\alpha_1} + 2^{-\gamma_0} + 2^{-\alpha_0} + |\mathcal{K}|^{-1} + 2^{-\lambda})\end{aligned}$$

Now in G_0, there can be at most one (partially) matching session for each session, and key confirmation tags produced in non-(partially-)matching $\textsc{Session}_{\mathsf{R}}$ and $\textsc{Session}_{\mathsf{I}}$ queries are different to each other. This implies that the adversary is unable to use key confirmation tags produced in initiator (respectively, responder) sessions to complete non-(partially-)matching responder (respectively, initiator) sessions.

GAME G_1. If $\mathcal{A}$ tests a type-1 or type-2 session, then the $\textsc{Test}$ oracle returns a random session key (cf. lines 34 to 35). We bound the probability difference of G_0 and G_1 in Lemma 4. Since the proof is straight-forward, for simplicity, we postpone the proof of Lemma 4 to Appendix E.3 and continue the proof of Theorem 5.

Lemma 4. *With the notations and assumptions from the proof of Theorem 5, there exists an adversary $\mathcal{B}$ that breaks the $(t', \varepsilon'_0, S, S)$-$\mathsf{MUC\text{-}IND\text{-}CCA}$-security of KEM_0 (that has key space $\mathcal{K}_0$ and error bound δ_0) with $t' \approx t$ and*

$$\left|\Pr[\mathsf{G}_0^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathsf{G}_1^{\mathcal{A}} \Rightarrow 1]\right| \leq 2\varepsilon'_0 + 2\mu S \cdot \delta_0 + \frac{2Q_{\mathsf{H}}\sqrt{S}}{\sqrt{|\mathcal{K}_0|}}.$$

GAME G_2. We add an flag $\mathsf{ForgeKC}_{\mathsf{R}}$ which captures the event that the adversary successfully forges a valid key-confirmation tag on behalf of some responder user r while r is uncorrupted and does not have such session. If this flag is raised to **true**, then the game aborts.

Concretely, we use a list $\mathcal{L}_{\mathtt{resp}}$ to record all key confirmation tags output by responder oracle $\textsc{Session}_{\mathsf{R}}$ (cf. lines 01 and 26). We raise a boolean flag $\mathsf{ForgeKC}_{\mathsf{R}}$ (which is initialized as **false** in line 02) to **true** if $\mathcal{A}$ sends a valid key-confirmation tag π to $\textsc{Der}_{\mathsf{I}}$ where π is not output by $\textsc{Session}_{\mathsf{R}}$ and the intended peer of this $\textsc{Der}_{\mathsf{I}}$ query is not corrupted at the time when $\textsc{Der}_{\mathsf{I}}$ receives π (cf. lines 51 to 53). Moreover, if $\mathsf{ForgeKC}_{\mathsf{R}}$ is raised to **true**, then the game aborts and returns a random bit.

GAMES G_0-G_4

00 $\mathsf{par}_1 \leftarrow \mathsf{Setup}_1(1^\lambda), \mathsf{par}_0 \leftarrow \mathsf{Setup}_0(1^\lambda)$
01 $\mathcal{L}_{\mathtt{resp}} := \varnothing$ //G_2-G_4
02 $\mathsf{ForgeKC}_{\mathtt{R}} :=$ **false** //G_2-G_4
03 $\mathcal{L}_{\mathtt{init}} := \varnothing$ //G_3-G_4
04 $\mathsf{ForgeKC}_{\mathtt{I}} :=$ **false** //G_3-G_4
05 **for** $n \in [\mu]$: $(\mathsf{pk}_n, \mathsf{sk}_n) \leftarrow \mathsf{Gen}_1(\mathsf{par}_1)$
06 $b \xleftarrow{\$} \{0,1\}$
07 $b' \leftarrow \mathcal{A}^{\mathrm{O}}(\mathsf{par}_1, \mathsf{par}_0, \mathsf{pk}_1, \cdots, \mathsf{pk}_\mu)$
08 **for** $\mathrm{sID}^* \in \mathcal{S}_{\mathtt{test}}$
09 **if** $\textsc{Fresh}(\mathrm{sID}^*) =$ **true**
10 **return** b //session not fresh
11 **if** $\textsc{Valid}(\mathrm{sID}^*) =$ **true**
12 **return** b //no valid attack
13 **return** $[\![b = b']\!]$

$\textsc{Session}_{\mathrm{R}}((i,r) \in [\mu]^2, m_{i,1})$
14 $\mathrm{cnt}_{\mathrm{S}}$ ++, $\mathrm{sID} := \mathrm{cnt}_{\mathrm{S}}$
15 $(\mathtt{Init}[\mathrm{sID}], \mathtt{Resp}[\mathrm{sID}]) := (i,r)$
16 $\mathtt{Type}[\mathrm{sID}] :=$ "Re"
17 $(\widetilde{\mathsf{pk}}, \mathsf{c}) := m_{i,1}$
18 $(\widetilde{\mathsf{c}}, \widetilde{\mathsf{k}}) \leftarrow \mathsf{Encaps}_0(\widetilde{\mathsf{pk}})$
19 $(\mathsf{c}_i, \mathsf{k}_i) \leftarrow \mathsf{Encaps}_1(\mathsf{pk}_i)$
20 $\mathsf{k}_r := \mathsf{Decaps}_1(\mathsf{sk}_r, \mathsf{c})$
21 $\mathrm{ctxt} := (\mathsf{pk}_i, \mathsf{pk}_r, \widetilde{\mathsf{pk}}, \widetilde{\mathsf{c}}, \mathsf{c}_i, \mathsf{c}_r)$
22 $\pi_r := \mathsf{G}_{\mathrm{R}}((\widetilde{\mathsf{k}}, \mathsf{k}_i, \mathsf{k}_r), \mathrm{ctxt})$
23 $\pi'_i := \mathsf{G}_{\mathrm{I}}((\widetilde{\mathsf{k}}, \mathsf{k}_i, \mathsf{k}_r), \mathrm{ctxt})$
24 $K' := \mathsf{H}((\widetilde{\mathsf{k}}, \mathsf{k}_i, \mathsf{k}_r), \mathrm{ctxt})$
25 $m_r := (\widetilde{\mathsf{c}}, \mathsf{c}_i, \pi_r)$, $\mathrm{st}_r := (\pi'_i, K')$
26 $\mathcal{L}_{\mathtt{resp}} := \mathcal{L}_{\mathtt{resp}} \cup \{\pi_r\}$ //G_2-G_4
27 $(\mathtt{Msg}_{\mathtt{I},1}[\mathrm{sID}], \mathtt{Msg}_{\mathrm{R}}[\mathrm{sID}]) := (m_{i,1}, m_r)$
28 $\mathrm{st}[\mathrm{sID}] := \mathrm{st}_r$
29 **return** (sID, m_r)

$\textsc{Test}(\mathrm{sID})$
30 **if** $\mathrm{sID} \in \mathcal{S}_{\mathtt{test}}$: **return** $\perp$ //already tested
31 **if** $\mathtt{SK}[\mathrm{sID}] = \perp$: **return** $\perp$
32 $\mathcal{S}_{\mathtt{test}} := \mathcal{S}_{\mathtt{test}} \cup \{\mathrm{sID}\}$
33 $K_0^* := \mathtt{SK}[\mathrm{sID}]$, $K_1^* \xleftarrow{\$} \mathcal{K}$
34 **if** sID is type 1 or 2 in Table 2 //G_1-G_4
35 $K_0^* \xleftarrow{\$} \mathcal{K}$ //G_1-G_4
36 **if** sID is type 5 or 6 in Table 2 //G_4
37 $K_0^* \xleftarrow{\$} \mathcal{K}$ //G_4
38 **return** K_b^*

$\textsc{Der}_{\mathrm{I}}(\mathrm{sID} \in [\mathrm{cnt}_{\mathrm{S}}], m_r)$
39 **if** $\mathtt{SK}[\mathrm{sID}] \neq \perp$ **or** $\mathtt{Type}[\mathrm{sID}] \neq$ "In"
40 **return** $\perp$ //no re-use
41 $(i,r) := (\mathtt{Init}[\mathrm{sID}], \mathtt{Resp}[\mathrm{sID}])$
42 $\mathrm{st}_i := \mathtt{ST}[\mathrm{sID}]$
43 $\mathtt{peerPreCor}[\mathrm{sID}] := \mathtt{cor}[r]$
44 $(\widetilde{\mathsf{c}}, \mathsf{c}, \pi) := m_r$
45 $\mathrm{st}_i := (\mathsf{pk}_r, \widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}, \mathsf{c}_r, \mathsf{k}_r)$
46 $\mathsf{k}_i := \mathsf{Decaps}_1(\mathsf{sk}_i, \mathsf{c}), \widetilde{\mathsf{k}} := \mathsf{Decaps}_0(\widetilde{\mathsf{sk}}, \widetilde{\mathsf{c}})$
47 $\mathrm{ctxt} := (\mathsf{pk}_i, \mathsf{pk}_r, \widetilde{\mathsf{pk}}, \widetilde{\mathsf{c}}, \mathsf{c}_i, \mathsf{c}_r)$
48 $\pi'_r := \mathsf{G}_{\mathrm{R}}((\widetilde{\mathsf{k}}, \mathsf{k}_i, \mathsf{k}_r), \mathrm{ctxt})$
49 $\pi_i := \mathsf{G}_{\mathrm{I}}((\widetilde{\mathsf{k}}, \mathsf{k}_i, \mathsf{k}_r), \mathrm{ctxt})$
50 **if** $\pi'_r = \pi$
51 **if** $\mathtt{peerPreCor}[\mathrm{sID}] =$ **false and** $\pi'_r \notin \mathcal{L}_{\mathtt{resp}}$ //G_2-G_4
52 $\mathsf{ForgeKC}_{\mathrm{R}} :=$ **true** //G_2-G_4
53 **abort** //G_2-G_4
54 $K := \mathsf{H}((\widetilde{\mathsf{k}}, \mathsf{k}_i, \mathsf{k}_r), \mathrm{ctxt})$
55 $\mathtt{SK}[\mathrm{sID}] := K$
56 **else**
57 $\mathtt{SK}[\mathrm{sID}] :=$ "reject"
58 **return** $\perp$
59 $m_{i,2} := \pi_i$
60 $\mathcal{L}_{\mathtt{init}} := \mathcal{L}_{\mathtt{init}} \cup \{\pi_i\}$ //G_3-G_4
61 **return** $m_{i,2}$

$\textsc{Der}_{\mathrm{R}}(\mathrm{sID} \in [\mathrm{cnt}_{\mathrm{S}}], m_{i,2})$
62 **if** $\mathtt{SK}[\mathrm{sID}] \neq \perp$ **or** $\mathtt{Type}[\mathrm{sID}] \neq$ "Re"
63 **return** $\perp$ //no re-use
64 $(i,r) := (\mathtt{Init}[\mathrm{sID}], \mathtt{Resp}[\mathrm{sID}])$
65 $\mathrm{st}_r := \mathtt{ST}[\mathrm{sID}]$
66 $\mathtt{peerPreCor}[\mathrm{sID}] := \mathtt{cor}[i]$
67 $\pi := m_{i,2}$
68 $(\pi', K') := \mathrm{st}_r$
69 **if** $\pi' = \pi$
70 **if** $\mathtt{peerPreCor}[\mathrm{sID}] =$ **false and** $\pi'_r \notin \mathcal{L}_{\mathtt{init}}$ //G_3-G_4
71 $\mathsf{ForgeKC}_{\mathrm{I}} :=$ **true** //G_3-G_4
72 **abort** //G_3-G_4
73 $K := K'$
74 $\mathtt{SK}[\mathrm{sID}] := K$
75 **else**
76 $\mathtt{SK}[\mathrm{sID}] :=$ "reject"
77 **return** $\perp$
78 $\mathtt{Msg}_{\mathtt{I},2}[\mathrm{sID}] := m_{i,2}$
79 **return** ε

Fig. 21. Games sequence in proving Theorem 5. $\mathcal{A}$ has access to oracles $\mathrm{O} := \{\textsc{Session}_{\mathrm{I}}, \textsc{Session}_{\mathrm{R}}, \textsc{Der}_{\mathrm{I}}, \textsc{Der}_{\mathrm{R}}, \textsc{Reveal}, \textsc{Corr}, \textsc{Test}, \mathsf{G}_{\mathrm{R}}, \mathsf{G}_{\mathrm{I}}, \mathsf{H}\}$, where $\mathcal{A}$ has quantum access to random oracles G_{R}, G_{I}, and H. Helper procedures Fresh and Valid are defined in Figure 3. A clean description of G_0 is given in Figure 24.

If $\mathsf{ForgeKC}_{\mathrm{R}} =$ **false**, then G_1 proceeds identically with G_2 and the winning probabilities of $\mathcal{A}$ in these two games are the same. Therefore, we have

$$\left|\Pr[\mathsf{G}_1^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathsf{G}_2^{\mathcal{A}} \Rightarrow 1]\right| \leq \Pr[\mathsf{ForgeKC}_{\mathrm{R}}],$$

where $\Pr[\mathsf{ForgeKC}_{\mathrm{R}}]$ is the probability that $\mathsf{ForgeKC}_{\mathrm{R}}$ is raised to **true** in G_1.

We use intermediate games $\mathsf{G}_{1,0}$-$\mathsf{G}_{1,2}$ (cf. Figure 22) to bound the probability of $\Pr[\mathsf{ForgeKC}_{\mathrm{R}}]$.

- Game $\mathsf{G}_{1,0}$. $\mathsf{G}_{1,0}$ proceeds identically with G_1 except that $\mathsf{G}_{1,0}$ outputs 1 if and only if the flag $\mathsf{ForgeKC}_{\mathrm{R}}$ is raised to **true** (cf. lines 08 and 57) in G_1. We have

$$\Pr[\mathsf{ForgeKC}_{\mathrm{R}} : \mathsf{G}_1^{\mathcal{A}}] = \Pr[\mathsf{G}_{1,0}^{\mathcal{A}} \Rightarrow 1].$$

- Game $\mathsf{G}_{1,1}$. In this game, we predict which responder user will be impersonated by the adversary $\mathcal{A}$ when $\mathcal{A}$ causes $\mathsf{ForgeKC}_{\mathrm{R}}$ to be **true**, and if we guess wrong, then the game aborts with output 0. Concretely, we guess a user r^* uniformly at random before running $\mathcal{A}$ (cf. line 00), and the game aborts

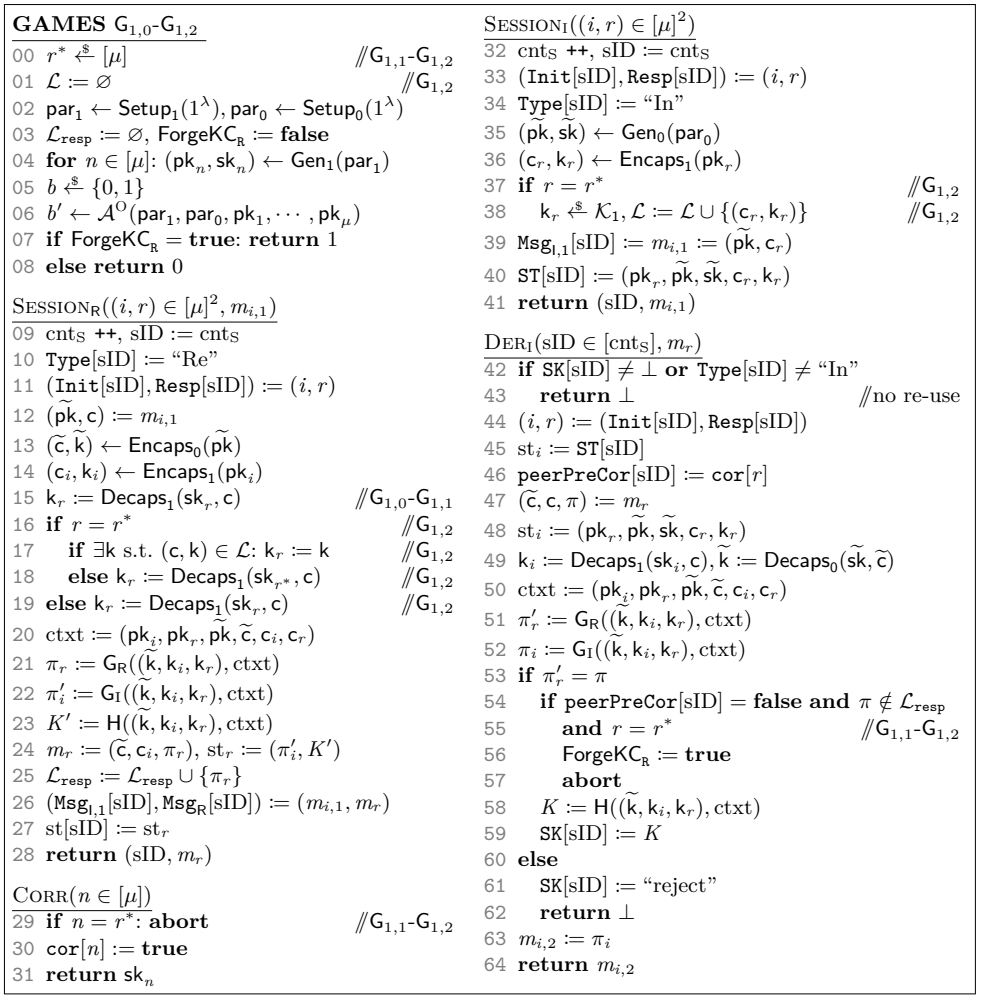

Fig. 22. Games sequence $\mathsf{G}_{1,0}$-$\mathsf{G}_{1,2}$ in bounding $\Pr[\mathsf{ForgeKC}_{\mathsf{R}}]$. Different to G_1, $\mathsf{G}_{1,0}$-$\mathsf{G}_{1,2}$ output 1 if and only if $\mathcal{A}$ causes $\mathsf{ForgeKC}_{\mathsf{R}}$ to be **true**. For simplicity, we only present the codes that different from G_1. All omitted oracles are the same as in G_1.

output 0 if $\mathcal{A}$ corrupts r^* (cf. line 29). $\mathsf{ForgeKC}_{\mathsf{R}}$ now is raised to **true** only if the responder that $\mathcal{A}$ impersonates is r^* (cf. line 55).

We claim that $\Pr[\mathsf{G}^{\mathcal{A}}_{1,1} \Rightarrow 1] = \frac{1}{\mu}\Pr[\mathsf{G}^{\mathcal{A}}_{1,0} \Rightarrow 1]$. Suppose that $\mathcal{A}$ corrupted in total μ' users in $\mathsf{G}_{1,0}$, where $1 \le \mu' < \mu$ is arbitrary, then the probability that $\mathsf{ForgeKC}_{\mathsf{R}}$ is raised to **true** (and thus the game $\mathsf{G}_{1,0}$ outputs 1) and r^* is the responder that $\mathcal{A}$ impersonates in triggering $\mathsf{ForgeKC}_{\mathsf{R}}$ is

$$\frac{\mu-\mu'}{\mu} \cdot \frac{1}{\mu-\mu'} \cdot \Pr[\mathsf{G}^{\mathcal{A}}_{1,0} \Rightarrow 1] = \frac{1}{\mu}\Pr[\mathsf{G}^{\mathcal{A}}_{1,0} \Rightarrow 1],$$

where $\frac{\mu-\mu'}{\mu}$ is the probability that r^* has not been corrupted yet and $\frac{1}{\mu-\mu'}$ is the probability that the responder that $\mathcal{A}$ impersonates in triggering $\mathsf{ForgeKC}_{\mathsf{R}}$ is r^* (note that r^* is sampled uniformly and independently at random). If $\mu' = \mu$, namely, $\mathcal{A}$ corrupts all users, then by definition of $\mathsf{ForgeKC}_{\mathsf{R}}$, $\mathcal{A}$ cannot trigger $\mathsf{ForgeKC}_{\mathsf{R}}$. By calculating all possibilities of μ', we have

$$\Pr[\mathsf{G}^{\mathcal{A}}_{1,1} \Rightarrow 1] = \frac{1}{\mu}\Pr[\mathsf{G}^{\mathcal{A}}_{1,0} \Rightarrow 1]$$

- GAME $\mathsf{G}_{1,2}$. In SESSION$_{\mathrm{I}}$, if the responder r is user r^*, then we sample k_r uniformly and independently at random instead of using Encaps_1 (cf. lines 37 to 38). We also use a list $\mathcal{L}$ (cf. line 01) that records all ciphertext-key pairs generated in SESSION$_{\mathrm{I}}$ (with responder r^*) to make the simulation consistent (cf. line 01 and lines 16 to 19). To bound the difference between $\mathsf{G}_{1,1}$ and $\mathsf{G}_{1,2}$, we construct a reduction $\mathcal{B}_1$ from MC-IND-CCA security of KEM_1 (cf. Figure 23).

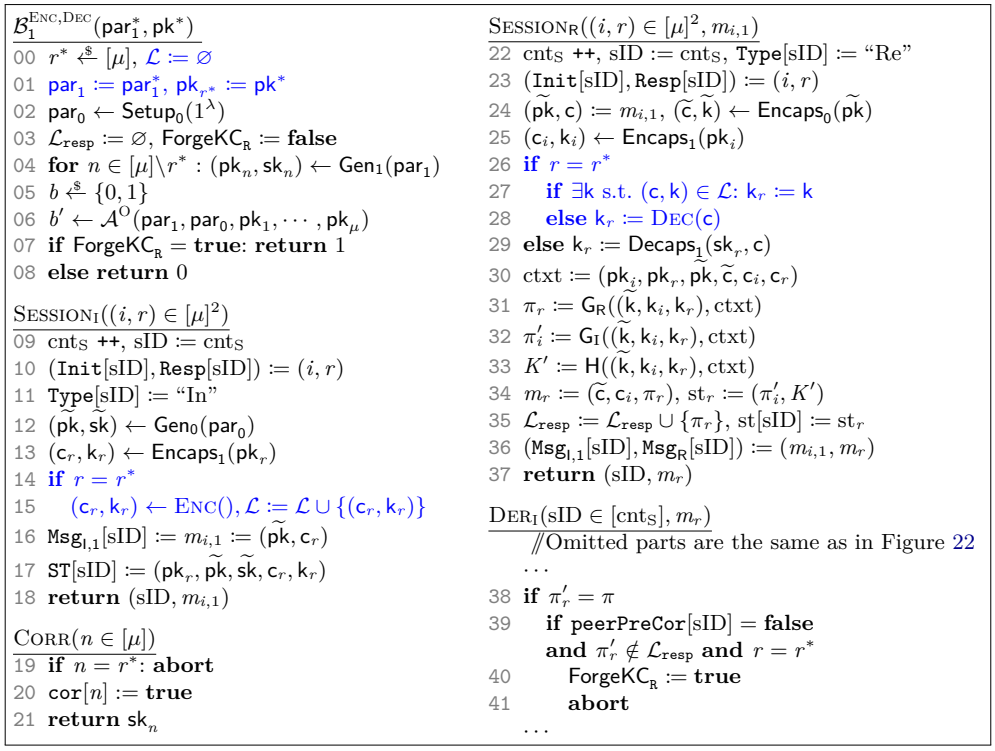

Fig. 23. Adversary $\mathcal{B}_1$ against MC-IND-CCA. $\mathcal{B}_1$ simulates $\mathsf{G}_{1,1}$ or $\mathsf{G}_{1,2}$ for $\mathcal{A}$. $\mathcal{B}_1$ has access to oracles Enc and Dec. For sake of simplicity, here we only present the codes of $\textsc{Session}_{\mathrm{I}}$ and $\textsc{Session}_{\mathrm{R}}$, other oracles are the same as in $\mathsf{G}_{1,1}$. Highlight lines show how we embed the challenge from MC-IND-CCA.

Reduction $\mathcal{B}_1$ plays the MC-IND-CCA game with at most S challenge ciphertexts. It firstly samples r^* uniformly at random, initializes a list $\mathcal{L}$ to record all challenge ciphertext-key pairs (cf. line 00), and set the public key of user r^* as the challenge public key. In $\textsc{Session}_{\mathrm{I}}$, if the responder is r^*, $\mathcal{B}_1$ embeds a challenge ciphertext-key pair into the protocol messages and records it in $\mathcal{L}$ (cf. line 38 and lines 14 to 15). In $\textsc{Session}_{\mathrm{R}}$, if the responder is r^*, $\mathcal{B}_1$ either uses the Dec oracle to decrypt the received ciphertext or finds the decapsulated key from $\mathcal{L}$, depends on whether the received ciphertext is challenge ciphertext (cf. lines 26 to 28).

If $\mathcal{B}_1$ plays $\mathsf{MC\text{-}IND\text{-}CCA}_{\mathsf{KEM}_1,0}$, then it perfectly simulates $\mathsf{G}_{1,1}$ for $\mathcal{A}$, and if it plays $\mathsf{MC\text{-}IND\text{-}CCA}_{\mathsf{KEM}_1,1}$, then it perfectly simulates $\mathsf{G}_{1,2}$. In line 15, we use the recorded KEM key without decryption, so we also need to consider the error bound δ_1 of KEM_1. Since there are at most S sessions in the AKE game, we have

$$\begin{aligned}
&\left|\Pr[\mathsf{G}_{1,1}^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathsf{G}_{1,2}^{\mathcal{A}} \Rightarrow 1]\right| \\
&\leq \left|\Pr[\mathsf{MC\text{-}IND\text{-}CCA}_{\mathsf{KEM}_1,0}^{\mathcal{B}_1} \Rightarrow 1] - \Pr[\mathsf{MC\text{-}IND\text{-}CCA}_{\mathsf{KEM}_1,1}^{\mathcal{B}_1} \Rightarrow 1]\right| + S \cdot \delta_1 \\
&\quad := \varepsilon'_1 + S\delta_1.
\end{aligned}$$

Now we bound $\Pr[\mathsf{G}_{1,2}^{\mathcal{A}} \Rightarrow 1]$. To makes $\mathsf{G}_{1,2}$ output 1, $\mathcal{A}$ needs to send a tag π to $\mathsf{Der}_{\mathrm{I}}$ such that (1) r^* is not corrupted yet when it sends π to $\mathsf{Der}_{\mathrm{I}}$, (2) π is not from $\textsc{Session}_{\mathrm{R}}$, namely, not recorded in $\mathcal{L}_{\mathtt{resp}}$, and (3) $\pi = \mathsf{G}_{\mathsf{R}}((\widetilde{\mathsf{k}}, \mathsf{k}_i, \mathsf{k}_{r^*}), \mathrm{ctxt})$ where k_{r^*} is uniformly random. The condition (2) requires that such tag π is forged by the adversary. To forge such π, the adversary needs to compute $\mathsf{G}_{\mathsf{R}}((\widetilde{\mathsf{k}}, \mathsf{k}_i, \mathsf{k}_{r^*}), \mathrm{ctxt})$ where k_{r^*} is uniformly random by condition (3).

We can use the argument in the proof of Lemma 4 (given in Appendix E.3) to prove that such $\mathsf{G}_{\mathsf{R}}((\widetilde{\mathsf{k}}, \mathsf{k}_i, \mathsf{k}_{r^*}), \mathrm{ctxt})$ are indistinguishable from independently and uniformly random tags except with probability $2Q_{\mathsf{G}_{\mathsf{R}}}\sqrt{S} \cdot (\sqrt{|\mathcal{K}_1|})^{-1}$ (since there are at most S session keys in the AKE game). Furthermore, if all tags are random, then the probability that $\mathcal{A}$ can forge at least one of these random tags is at most

$S/2^\lambda$ (similar to the event RandKC in the proof of Lemma 1). Therefore, we have

$$\Pr[\mathsf{G}^{\mathcal{A}}_{1,2} \Rightarrow 1] \leq \frac{2Q_{\mathsf{G}_\mathsf{R}}\sqrt{S}}{\sqrt{|\mathcal{K}_1|}} + \frac{S}{2^\lambda},$$

and by combining all probability differences, we have

$$\left|\Pr[\mathsf{G}^{\mathcal{A}}_1 \Rightarrow 1] - \Pr[\mathsf{G}^{\mathcal{A}}_2 \Rightarrow 1]\right| \leq \Pr[\mathsf{ForgeKC}_\mathsf{R}] \leq \mu(\varepsilon'_1 + S\delta_1 + \frac{2Q_{\mathsf{G}_\mathsf{R}}\sqrt{S}}{\sqrt{|\mathcal{K}_1|}} + \frac{S}{2^\lambda}) \ .$$

GAME G_3. We add an flag $\mathsf{ForgeKC}_\mathsf{I}$ which captures the event that the adversary successfully forges a valid key-confirmation tag on behalf of some initiator user i while user i is uncorrupted and does not have such session. If $\mathcal{A}$ triggers $\mathsf{ForgeKC}_\mathsf{I}$, then the game aborts and outputs a random bit.

Concretely, we use a list $\mathcal{L}_{\mathtt{init}}$ to record all key confirmation tags output by responder oracle $\textsc{Der}_\mathrm{I}$ (cf. lines 03 and 60). We raise the boolean flag $\mathsf{ForgeKC}_\mathsf{I}$ (which is initialized as **false** in line 04) to **true** if $\mathcal{A}$ sends a valid key-confirmation tag π to $\textsc{Der}_\mathsf{R}$ where π is not output by $\textsc{Der}_\mathsf{R}$ and the initiator party of this $\textsc{Der}_\mathsf{R}$ query is not corrupted at the time when $\textsc{Der}_\mathsf{R}$ receives π (cf. lines 70 to 72). If $\mathsf{ForgeKC}_\mathsf{I}$ is raised to **true**, then the game aborts and returns a random bit.

If $\mathsf{ForgeKC}_\mathsf{I} =$ **false**, then G_2 proceeds identically with G_3 and the winning probabilities of $\mathcal{A}$ in these two games are the same. Therefore, we have

$$\left|\Pr[\mathsf{G}^{\mathcal{A}}_2 \Rightarrow 1] - \Pr[\mathsf{G}^{\mathcal{A}}_3 \Rightarrow 1]\right| \leq \Pr[\mathsf{ForgeKC}_\mathsf{I}],$$

where $\Pr[\mathsf{ForgeKC}_\mathsf{I}]$ is the probability that $\mathsf{ForgeKC}_\mathsf{I}$ is raised to **true** in G_2.

Bounding $\Pr[\mathsf{ForgeKC}_\mathsf{I}]$ is similar to bounding $\Pr[\mathsf{ForgeKC}_\mathsf{R}]$ since these two events are symmetric. For sake of simplicity, we leave it as a lemma (cf. Lemma 5) and prove it in Appendix E.3.

Lemma 5. *With the notations and assumptions from the proof of Theorem 5, there exists an adversary $\mathcal{B}_1$ that breaks the (t', ε'_1, S)-$\mathsf{MC\text{-}IND\text{-}CCA}$-security of KEM_1 (that has key space $\mathcal{K}_1$ and error bound δ_1) with $t' \approx t$ and*

$$\left|\Pr[\mathsf{G}^{\mathcal{A}}_2 \Rightarrow 1] - \Pr[\mathsf{G}^{\mathcal{A}}_3 \Rightarrow 1]\right| \leq \Pr[\mathsf{ForgeKC}_\mathsf{I}] \leq \mu(\varepsilon'_1 + S\delta_1 + \frac{2Q_{\mathsf{G}_\mathsf{I}}\sqrt{S}}{\sqrt{|\mathcal{K}_1|}} + \frac{S}{2^\lambda}).$$

GAME G_4. If $\mathcal{A}$ tests a session sID that is of type 5 or 6, then the $\textsc{Test}$ oracle returns a random session key (cf. lines 36 to 37). Now in G_4, all session keys output by $\textsc{Test}$ are independently and uniformly random, thus $\mathcal{A}$ does not have have any distinguishing advantage. Therefore,

$$\Pr[\mathsf{G}^{\mathcal{A}}_4 \Rightarrow 1] = \frac{1}{2} \ .$$

We claim that $\Pr[\mathsf{G}^{\mathcal{A}}_3 \Rightarrow 1] = \Pr[\mathsf{G}^{\mathcal{A}}_4 \Rightarrow 1]$. In fact, the abort events introduced in G_2 and G_3 make $\mathcal{A}$ unable to test any type-5 or type-6 session, and thus the modification of returned session keys of type-5 and type-6 sessions do not change $\mathcal{A}$'s view.

To see this, let sID be an arbitrary type-5 session. Let $i =$ owner[sID]. By definition (cf. Table 2), sID is an initiator session (i.e., $\mathsf{Type}[\mathrm{sID}] =$ "In"), so $\mathcal{A}$ acts as some responder r of user i in sID.

To complete session sID, $\mathcal{A}$ firstly activates sID via querying $\textsc{Session}_\mathrm{I}((i, r))$ (which responses $(\mathrm{sID}, (\widetilde{\mathsf{pk}}, \mathsf{c}_r))$) and then queries $\textsc{Der}_\mathrm{I}(\mathrm{sID}, (\widetilde{\mathsf{c}}, \mathsf{c}_i, \pi))$. By the definition of type-5 session (cf. Table 2), sID does not have partially matching session, which implies that such π is not from any $\textsc{Session}_\mathsf{R}(i, r)$ query. By the definition of type-5 session, r is not corrupted at the time $\mathcal{A}$ queries $\textsc{Der}_\mathrm{I}(\mathrm{sID}, (\widetilde{\mathsf{c}}, \mathsf{c}_i, \pi))$. Therefore, if such $\textsc{Der}_\mathrm{I}$ does not reply $\perp$, then $\mathcal{A}$ produces the correct key confirmation tag π_r of this session. By lines 51 to 53, this will trigger $\mathsf{ForgeKC}_\mathsf{R}$ and make the game abort. Therefore, $\mathcal{A}$ cannot test any type-5 session.

Similarly, $\mathcal{A}$ cannot test any type-6 session. Let sID be an arbitrary type-6 session. This case is symmetric to the type-5 case. To complete sID, $\mathcal{A}$ has to produce the correct key confirmation tag π_i of this session. By the definition of type-6 session (cf. Table 2), such tag is not from any initiator session and the initiator user of sID is not corrupted at the time $\mathcal{A}$ sends π_i to $\textsc{Der}_\mathsf{R}$. Therefore, if $\mathcal{A}$ produces such correct key confirmation tag, then it triggers $\mathsf{ForgeKC}_\mathsf{R}$ and makes the game abort.

Combining all the probability differences in the games sequence G_0-G_4, we have

$$\varepsilon = |\Pr[\mathsf{IND\text{-}FS}^{\mathcal{A}}_{\mathsf{AKE}_{\mathsf{kem}}} \Rightarrow 1] - \frac{1}{2}| \leq 2\varepsilon'_0 + 2\mu\varepsilon'_1 + 2\mu S(\delta_0 + \delta_1) + \mu^2 2^{-\gamma_1} + \mu S 2^{-\lambda+1} + S^2 \cdot (2^{-\alpha_1} + 2^{-\gamma_0} + 2^{-\alpha_0}) + \frac{2\mu(Q_{\mathsf{G_R}} + Q_{\mathsf{G_I}})\sqrt{S}}{\sqrt{|\mathcal{K}_1|}} + \frac{2Q_{\mathsf{H}}\sqrt{S}}{\sqrt{|\mathcal{K}_0|}}$$

GAME G_0

00 $\mathsf{par}_1 \leftarrow \mathsf{Setup}_1(1^\lambda), \mathsf{par}_0 \leftarrow \mathsf{Setup}_0(1^\lambda)$
01 **for** $n \in [\mu]$
02 $\quad(\mathsf{pk}_n, \mathsf{sk}_n) \leftarrow \mathsf{Gen}_1(\mathsf{par}_1)$
03 $b \xleftarrow{\$} \{0,1\}$
04 $b' \leftarrow \mathcal{A}^{\mathrm{O}}(\mathsf{par}_1, \mathsf{par}_0, \mathsf{pk}_1, \cdots, \mathsf{pk}_\mu)$
05 **for** $\mathrm{sID}^* \in \mathcal{S}_{\mathtt{test}}$
06 $\quad$**if** $\textsc{Fresh}(\mathrm{sID}^*) = \mathbf{true}$
07 $\quad\quad$**return** b //session not fresh
08 $\quad$**if** $\textsc{Valid}(\mathrm{sID}^*) = \mathbf{true}$
09 $\quad\quad$**return** b //no valid attack
10 **return** $[\![b = b']\!]$

$\textsc{Session}_{\mathrm{R}}((i,r) \in [\mu]^2, m_{i,1})$
11 $\mathrm{cnt}_{\mathrm{S}}$ ++
12 $\mathrm{sID} := \mathrm{cnt}_{\mathrm{S}}$
13 $(\mathtt{Init}[\mathrm{sID}], \mathtt{Resp}[\mathrm{sID}]) := (i,r)$
14 $\mathtt{Type}[\mathrm{sID}] :=$ "Re"
15 $(\widetilde{\mathsf{pk}}, \mathsf{c}) := m_{i,1}$
16 $(\widetilde{\mathsf{c}}, \widetilde{\mathsf{k}}) \leftarrow \mathsf{Encaps}_0(\widetilde{\mathsf{pk}})$
17 $(\mathsf{c}_i, \mathsf{k}_i) \leftarrow \mathsf{Encaps}_1(\mathsf{pk}_i)$
18 $\mathsf{k}_r := \mathsf{Decaps}_1(\mathsf{sk}_r, \mathsf{c})$
19 $\mathrm{ctxt} := (\mathsf{pk}_i, \mathsf{pk}_r, \widetilde{\mathsf{pk}}, \widetilde{\mathsf{c}}, \mathsf{c}_i, \mathsf{c}_r)$
20 $\pi_r := \mathsf{G_R}((\widetilde{\mathsf{k}}, \mathsf{k}_i, \mathsf{k}_r), \mathrm{ctxt})$
21 $\pi'_i := \mathsf{G_I}((\widetilde{\mathsf{k}}, \mathsf{k}_i, \mathsf{k}_r), \mathrm{ctxt})$
22 $K' := \mathsf{H}((\widetilde{\mathsf{k}}, \mathsf{k}_i, \mathsf{k}_r), \mathrm{ctxt})$
23 $m_r := (\widetilde{\mathsf{c}}, \mathsf{c}_i, \pi_r)$, $\mathrm{st}_r := (\pi'_i, K')$
24 $(\mathtt{Msg}_{\mathsf{l},1}[\mathrm{sID}], \mathtt{Msg}_{\mathsf{R}}[\mathrm{sID}]) := (m_{i,1}, m_r)$
25 $\mathrm{st}[\mathrm{sID}] := \mathrm{st}_r$
26 **return** (sID, m_r)

$\textsc{Der}_{\mathrm{R}}(\mathrm{sID} \in [\mathrm{cnt}_{\mathrm{S}}], m_{i,2})$
27 **if** $\mathtt{SK}[\mathrm{sID}] \neq \perp$ **or** $\mathtt{Type}[\mathrm{sID}] \neq$ "Re"
28 $\quad$**return** $\perp$ //no re-use
29 $(i,r) := (\mathtt{Init}[\mathrm{sID}], \mathtt{Resp}[\mathrm{sID}])$
30 $\mathrm{st}_r := \mathtt{ST}[\mathrm{sID}]$
31 $\mathtt{peerPreCor}[\mathrm{sID}] := \mathtt{cor}[i]$
32 $\pi := m_{i,2}$, $(\pi', K') := \mathrm{st}_r$
33 **if** $\pi' = \pi$
34 $\quad K := K'$
35 $\quad\mathtt{SK}[\mathrm{sID}] := K$
36 **else**
37 $\quad\mathtt{SK}[\mathrm{sID}] :=$ "reject"
38 $\quad$**return** $\perp$
39 $\mathtt{Msg}_{\mathsf{l},2}[\mathrm{sID}] := m_{i,2}$
40 **return** ε

$\textsc{Session}_{\mathrm{I}}((i,r) \in [\mu]^2)$
41 $\mathrm{cnt}_{\mathrm{S}}$ ++, $\mathrm{sID} := \mathrm{cnt}_{\mathrm{S}}$
42 $(\mathtt{Init}[\mathrm{sID}], \mathtt{Resp}[\mathrm{sID}]) := (i,r)$
43 $\mathtt{Type}[\mathrm{sID}] :=$ "In"
44 $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}) \leftarrow \mathsf{Gen}_0(\mathsf{par}_0)$
45 $(\mathsf{c}_r, \mathsf{k}_r) \leftarrow \mathsf{Encaps}_1(\mathsf{pk}_r)$
46 $\mathtt{Msg}_{\mathsf{l},1}[\mathrm{sID}] := m_{i,1} := (\widetilde{\mathsf{pk}}, \mathsf{c}_r)$
47 $\mathtt{ST}[\mathrm{sID}] := (\mathsf{pk}_r, \widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}, \mathsf{c}_r, \mathsf{k}_r)$
48 **return** $(\mathrm{sID}, m_{i,1})$

$\textsc{Der}_{\mathrm{I}}(\mathrm{sID} \in [\mathrm{cnt}_{\mathrm{S}}], m_r)$
49 **if** $\mathtt{SK}[\mathrm{sID}] \neq \perp$ **or** $\mathtt{Type}[\mathrm{sID}] \neq$ "In"
50 $\quad$**return** $\perp$ //no re-use
51 $(i,r) := (\mathtt{Init}[\mathrm{sID}], \mathtt{Resp}[\mathrm{sID}])$
52 $\mathrm{st}_i := \mathtt{ST}[\mathrm{sID}]$
53 $\mathtt{peerPreCor}[\mathrm{sID}] := \mathtt{cor}[r]$
54 $(\widetilde{\mathsf{c}}, \mathsf{c}, \pi) := m_r$
55 $\mathrm{st}_i := (\mathsf{pk}_r, \widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}, \mathsf{c}_r, \mathsf{k}_r)$
56 $\mathsf{k}_i := \mathsf{Decaps}_1(\mathsf{sk}_i, \mathsf{c})$
57 $\widetilde{\mathsf{k}} := \mathsf{Decaps}_0(\widetilde{\mathsf{sk}}, \widetilde{\mathsf{c}})$
58 $\mathrm{ctxt} := (\mathsf{pk}_i, \mathsf{pk}_r, \widetilde{\mathsf{pk}}, \widetilde{\mathsf{c}}, \mathsf{c}_i, \mathsf{c}_r)$
59 $\pi'_r := \mathsf{G_R}((\widetilde{\mathsf{k}}, \mathsf{k}_i, \mathsf{k}_r), \mathrm{ctxt})$
60 $\pi_i := \mathsf{G_I}((\widetilde{\mathsf{k}}, \mathsf{k}_i, \mathsf{k}_r), \mathrm{ctxt})$
61 **if** $\pi'_r = \pi$
62 $\quad K := \mathsf{H}((\widetilde{\mathsf{k}}, \mathsf{k}_i, \mathsf{k}_r), \mathrm{ctxt})$
63 $\quad\mathtt{SK}[\mathrm{sID}] := K$
64 **else**
65 $\quad\mathtt{SK}[\mathrm{sID}] :=$ "reject"
66 $\quad$**return** $\perp$
67 $m_{i,2} := \pi_i$
68 **return** $m_{i,2}$

$\textsc{Reveal}(\mathrm{sID})$
69 $\mathtt{revSK}[\mathrm{sID}] := \mathbf{true}$
70 **return** $\mathtt{SK}[\mathrm{sID}]$

$\textsc{Corr}(n \in [\mu])$
71 $\mathtt{cor}[n] := \mathbf{true}$
72 **return** sk_n

$\textsc{Test}(\mathrm{sID})$
73 **if** $\mathrm{sID} \in \mathcal{S}_{\mathtt{test}}$ **return** $\perp$ //already tested
74 **if** $\mathtt{SK}[\mathrm{sID}] = \perp$ **return** $\perp$
75 $\mathcal{S}_{\mathtt{test}} := \mathcal{S}_{\mathtt{test}} \cup \{\mathrm{sID}\}$
76 $K_0^* := \mathtt{SK}[\mathrm{sID}]$
77 $K_1^* \xleftarrow{\$} \mathcal{K}$
78 **return** K_b^*

Fig. 24. Full codes of G_0 in proving Theorem 5. $\mathcal{A}$ has access to oracles $\mathrm{O} := \{\textsc{Session}_{\mathrm{I}}, \textsc{Session}_{\mathrm{R}}, \textsc{Der}_{\mathrm{I}}, \textsc{Der}_{\mathrm{R}}, \textsc{Reveal}, \textsc{Corr}, \textsc{Test}, \mathsf{G_R}, \mathsf{G_I}, \mathsf{H}\}$, where $\mathcal{A}$ has quantum access to random oracles $\mathsf{G_R}$, $\mathsf{G_I}$, and H. FRESH and VALID are defined in Figure 3.

E.3 Postponed Proofs of Lemmas 4 and 5

Proof (Lemma 4). To make the games transition more readable, we use intermediate games $\mathsf{G}_{0,0}$-$\mathsf{G}_{0,3}$ to show the transition step by step. The codes of $\mathsf{G}_{0,0}$-$\mathsf{G}_{0,3}$ is shown in Figure 25.

- Game $\mathsf{G}_{0,0}$ and $\mathsf{G}_{0,3}$. In these games, we use a list $\widetilde{\mathcal{L}}$ to record all public keys, ciphertexts, and KEM keys of KEM_0 that are generated in oracles: (1) All public keys for KEM_0 generated in $\textsc{Session}_{\mathrm{I}}$ are recorded in $\widetilde{\mathcal{L}}$ (cf. line 42). (2) In $\textsc{Session}_{\mathsf{R}}$, if the received public key $\widetilde{\mathsf{pk}}$ for KEM_0 is recorded in $\widetilde{\mathcal{L}}$, then the simulator also records the ciphertext and KEM key encapsulated by $\widetilde{\mathsf{pk}}$ in $\widetilde{\mathcal{L}}$ (cf. lines 18 to 22). (3) In $\textsc{Der}_{\mathrm{I}}$, if the received ciphertext for KEM_0 is recorded in $\widetilde{\mathcal{L}}$, then the simulator will use the key recorded in $\widetilde{\mathcal{L}}$ as the decapsulated key (cf. lines 53 to 55).

These modifications do not change the view of $\mathcal{A}$ unless there exist at least one of ciphertexts generated in $\textsc{Der}_{\mathrm{I}}$ cause decryption error of KEM_0 (cf. Definition 6). We have

$$\left|\Pr[\mathsf{G}_0^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathsf{G}_{0,0}^{\mathcal{A}} \Rightarrow 1]\right| \leq S \cdot \delta_0$$
$$\left|\Pr[\mathsf{G}_{0,3}^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathsf{G}_1^{\mathcal{A}} \Rightarrow 1]\right| \leq S \cdot \delta_0$$

- Game $\mathsf{G}_{0,1}$ and $\mathsf{G}_{0,2}$. Compared with games $\mathsf{G}_{0,0}$ and $\mathsf{G}_{0,3}$, respectively, $\mathsf{G}_{0,1}$ and $\mathsf{G}_{0,2}$ use independent random keys as the encapsulated KEM key in $\textsc{Session}_{\mathsf{R}}$ (cf. line 20). To tell the difference between $\mathsf{G}_{0,0}$ and $\mathsf{G}_{0,1}$ (and similarly, $\mathsf{G}_{0,2}$ and $\mathsf{G}_{0,3}$), the adversary needs to distinguish if the encapsulated KEM keys generated in $\textsc{Session}_{\mathsf{R}}$ are real or random. We bound the differences between $\mathsf{G}_{0,0}$ and $\mathsf{G}_{0,1}$ by a direction reduction from KEM_0.

We construct a reduction $\mathcal{B}_0$ that simulates $\mathsf{G}_{0,0}$ or $\mathsf{G}_{0,1}$ for $\mathcal{A}$ to attack the MUC-IND-CCA security of KEM_0. $\mathcal{B}_0$ works as follows: It plays the $\mathsf{MUC\text{-}IND\text{-}CCA}$ game with S users and at most S challenge ciphertexts per users (S is the number of session in the AKE game). By Definition 9, $\mathcal{B}_0$ has oracle access to $\textsc{Enc}$ and $\textsc{Dec}$. $\mathcal{B}_0$ embeds the challenge public keys in $\textsc{Session}_{\mathrm{I}}$, embeds the challenge ciphertexts (output by $\textsc{Enc}$) in $\textsc{Session}_{\mathsf{R}}$, and maintains a list $\widetilde{\mathcal{L}}$ which records all embedded challenges. In $\textsc{Session}'_{\mathrm{I}}$, if the receiving ciphertext $\widetilde{\mathsf{c}}$ is not generated from $\textsc{Session}_{\mathsf{R}}$ (which means that it is not challenge ciphertext), then $\mathcal{B}_0$ uses $\textsc{Dec}$ to decrypt it. The full description of $\mathcal{B}_0$ is given in Figure 26.

If $\mathcal{B}_0$ plays $\mathsf{MUC\text{-}IND\text{-}CCA}_{\mathsf{KEM}_0,b}$, then it perfectly simulates $\mathsf{G}_{0,b}$ for $\mathcal{A}$. Therefore, we have

$$\begin{aligned}&\left|\Pr[\mathsf{G}_{0,0}^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathsf{G}_{0,1}^{\mathcal{A}} \Rightarrow 1]\right| \\ &\leq \left|\Pr[\mathsf{MUC\text{-}IND\text{-}CCA}_{\mathsf{KEM}_0,0}^{\mathcal{B}_0} \Rightarrow 1] - \Pr[\mathsf{MUC\text{-}IND\text{-}CCA}_{\mathsf{KEM}_0,1}^{\mathcal{B}_0} \Rightarrow 1]\right| =: \varepsilon'_0.\end{aligned}$$

A similar argument applies when bounding $\mathsf{G}_{0,2}$ and $\mathsf{G}_{0,3}$. So, we also have

$$\left|\Pr[\mathsf{G}_{0,2}^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathsf{G}_{0,3}^{\mathcal{A}} \Rightarrow 1]\right| \leq \varepsilon'_0,$$

where ε'_0 is the advantage to break MUC-IND-CCA security of KEM_0.

Now we bound the probability difference between $\mathsf{G}_{0,1}$ and $\mathsf{G}_{0,2}$. Let sID^* be arbitrary type-1 or type-2 (cf. Table 2) tested session. Let $\widetilde{\mathsf{k}}^*$ be the KEM key of KEM_0 in the hash input of $\mathtt{SK}[\mathrm{sID}^*]$. To distinguish $\mathsf{G}_{0,1}$ and $\mathsf{G}_{0,2}$, $\mathcal{A}$ needs to distinguish if the output of $\textsc{Test}(\mathrm{sID}^*)$ is the same as $\mathtt{SK}[\mathrm{sID}^*]$. That is, $\mathcal{A}$ needs to distinguish if $\textsc{Test}(\mathrm{sID}^*) \xleftarrow{\$} \mathcal{K}$ or $\textsc{Test}(\mathrm{sID}^*) = \mathsf{H}((\widetilde{\mathsf{k}}^*, \mathsf{k}_i^*, \mathsf{k}_r^*), \mathrm{ctxt}^*)$, where $((\widetilde{\mathsf{k}}^*, \mathsf{k}_i^*, \mathsf{k}_r^*), \mathrm{ctxt}^*)$ is the hash input of $\mathtt{SK}[\mathrm{sID}^*]$.

We first claim that, in $\mathsf{G}_{0,1}$, if a tested session sID^* is type-1 or type-2 in Table 2, then the KEM key for KEM_0, $\widetilde{\mathsf{k}}^*$, of the hash input of $\mathtt{SK}[\mathrm{sID}^*]$ is independently and uniformly at random. To see this, suppose that sID^* is type-1, then by definition, $\mathtt{SK}[\mathrm{sID}^*]$ has a partially matching session, which means that the KEM ciphertext of KEM_0 received by sID is generated by $\textsc{Session}_{\mathsf{R}}$. By the codes of $\mathsf{G}_{0,1}$ (cf. line 54), the KEM key of KEM_0 in session sID is uniformly random. A similar argument also applies to the case of type-2.

Now we are ready to use Lemma 3 to bound the distinguishing probability of $\mathcal{A}$. We observed that H in $\mathsf{G}_{0,1}$ differs H in $\mathsf{G}_{0,2}$ on the set of hash inputs of type-1 or type-2 tested sessions, and this set can be viewed as uniformly at random over $\mathcal{K}_0$ since their $\widetilde{\mathsf{k}}^*$ value is uniformly at random, as we claimed before. To use Lemma 3, H in $\mathsf{G}_{0,1}$ and H in $\mathsf{G}_{0,2}$ are viewed as qRO_0 and qRO_1 in Lemma 3, respectively,

GAMES $\mathsf{G}_0, \mathsf{G}_{0,0}$-$\mathsf{G}_{0,3}, \mathsf{G}_1$

$\mathsf{par}_1 \leftarrow \mathsf{Setup}_1(1^\lambda), \mathsf{par}_0 \leftarrow \mathsf{Setup}_0(1^\lambda)$
for $n \in [\mu]$
  $(\mathsf{pk}_n, \mathsf{sk}_n) \leftarrow \mathsf{Gen}_1(\mathsf{par}_1)$
$b \xleftarrow{\$} \{0,1\}$
$\widetilde{\mathcal{L}} := \{\}$ //$\mathsf{G}_{0,0}$-$\mathsf{G}_{0,3}$
$b' \leftarrow \mathcal{A}^{\mathrm{O}}(\mathsf{par}_1, \mathsf{par}_0, \mathsf{pk}_1, \cdots, \mathsf{pk}_\mu)$
for $\mathrm{sID}^* \in \mathcal{S}_{\mathtt{test}}$
  if $\textsc{Fresh}(\mathrm{sID}^*) = \mathbf{false}$
    return b //session not fresh
  if $\textsc{Valid}(\mathrm{sID}^*) = \mathbf{false}$
    return b //no valid attack
return $[\![b = b']\!]$

$\textsc{Session}_{\mathrm{R}}((i,r) \in [\mu]^2, m_{i,1})$

$\mathrm{cnt}_{\mathrm{S}}$ ++
$\mathrm{sID} := \mathrm{cnt}_{\mathrm{S}}$
$(\mathtt{Init}[\mathrm{sID}], \mathtt{Resp}[\mathrm{sID}]) := (i,r)$
$\mathtt{Type}[\mathrm{sID}] :=$ "Re"
$(\widetilde{\mathsf{pk}}, \mathsf{c}) := m_{i,1}$
$(\widetilde{\mathsf{c}}, \widetilde{\mathsf{k}}) \leftarrow \mathsf{Encaps}_0(\widetilde{\mathsf{pk}})$ //G_0,G_1
if $(\widetilde{\mathsf{pk}}, (\bot, \bot)) \in \widetilde{\mathcal{L}}$ //$\mathsf{G}_{0,0}$-$\mathsf{G}_{0,3}$
  $(\widetilde{\mathsf{c}}, \widetilde{\mathsf{k}}) \leftarrow \mathsf{Encaps}_0(\widetilde{\mathsf{pk}})$ //$\mathsf{G}_{0,0}$-$\mathsf{G}_{0,3}$
  $\widetilde{\mathsf{k}} \xleftarrow{\$} \mathcal{K}_0$ //$\mathsf{G}_{0,1}$-$\mathsf{G}_{0,2}$
  $\widetilde{\mathcal{L}} := \widetilde{\mathcal{L}} \cup \{(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{c}}, \widetilde{\mathsf{k}})\}$ //$\mathsf{G}_{0,0}$-$\mathsf{G}_{0,3}$
else $(\widetilde{\mathsf{c}}, \widetilde{\mathsf{k}}) \leftarrow \mathsf{Encaps}_0(\widetilde{\mathsf{pk}})$ //$\mathsf{G}_{0,0}$-$\mathsf{G}_{0,3}$
$(\mathsf{c}_i, \mathsf{k}_i) \leftarrow \mathsf{Encaps}_1(\mathsf{pk}_i)$
$\mathsf{k}_r := \mathsf{Decaps}_1(\mathsf{sk}_r, \mathsf{c})$
$\mathrm{ctxt} := (\mathsf{pk}_i, \mathsf{pk}_r, \widetilde{\mathsf{pk}}, \widetilde{\mathsf{c}}, \mathsf{c}_i, \mathsf{c}_r)$
$\pi_r := \mathsf{G}_{\mathrm{R}}((\widetilde{\mathsf{k}}, \mathsf{k}_i, \mathsf{k}_r), \mathrm{ctxt})$
$\pi'_i := \mathsf{G}_{\mathrm{I}}((\widetilde{\mathsf{k}}, \mathsf{k}_i, \mathsf{k}_r), \mathrm{ctxt})$
$K' := \mathsf{H}((\widetilde{\mathsf{k}}, \mathsf{k}_i, \mathsf{k}_r), \mathrm{ctxt})$
$K' := \mathsf{H}((\widetilde{\mathsf{k}}, \mathsf{k}_i, \mathsf{k}_r), \mathrm{ctxt})$
$m_r := (\widetilde{\mathsf{c}}, \mathsf{c}_i)$
$\mathrm{st}_r := (\pi'_i, K')$
$(\mathtt{Msg}_{\mathsf{l},1}[\mathrm{sID}], \mathtt{Msg}_{\mathrm{R}}[\mathrm{sID}]) := (m_{i,1}, m_r)$
$\mathrm{st}[\mathrm{sID}] := \mathrm{st}_r$
return (sID, m_r)

$\textsc{Session}_{\mathrm{I}}((i,r) \in [\mu]^2)$

$\mathrm{cnt}_{\mathrm{S}}$ ++, $\mathrm{sID} := \mathrm{cnt}_{\mathrm{S}}$
$(\mathtt{Init}[\mathrm{sID}], \mathtt{Resp}[\mathrm{sID}]) := (i,r)$
$\mathtt{Type}[\mathrm{sID}] :=$ "In"
$(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}) \leftarrow \mathsf{Gen}_0(\mathsf{par}_0)$
$(\mathsf{c}_r, \mathsf{k}_r) \leftarrow \mathsf{Encaps}_1(\mathsf{pk}_r)$
$\mathtt{Msg}_{\mathsf{l},1}[\mathrm{sID}] := m_{i,1} := (\widetilde{\mathsf{pk}}, \mathsf{c}_r)$
$\mathtt{ST}[\mathrm{sID}] := (\mathsf{pk}_r, \widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}, \mathsf{c}_r, \mathsf{k}_r)$
$\widetilde{\mathcal{L}} := \widetilde{\mathcal{L}} \cup \{(\widetilde{\mathsf{pk}}, (\bot, \bot))\}$ //$\mathsf{G}_{0,0}$-$\mathsf{G}_{0,3}$
return $(\mathrm{sID}, m_{i,1})$

$\textsc{Der}_{\mathrm{I}}(\mathrm{sID} \in [\mathrm{cnt}_{\mathrm{S}}], m_r)$

if $\mathtt{SK}[\mathrm{sID}] \neq \bot$ **or** $\mathtt{Type}[\mathrm{sID}] \neq$ "In"
  return $\bot$ //no re-use
$(i,r) := (\mathtt{Init}[\mathrm{sID}], \mathtt{Resp}[\mathrm{sID}])$
$\mathrm{st}_i := \mathtt{ST}[\mathrm{sID}]$
$\mathtt{peerPreCor}[\mathrm{sID}] := \mathtt{cor}[r]$
$(\widetilde{\mathsf{c}}, \mathsf{c}, \pi) := m_r$
$\mathrm{st}_i := (\mathsf{pk}_r, \widetilde{\mathsf{pk}}, \widetilde{\mathsf{sk}}, \mathsf{c}_r, \mathsf{k}_r)$
$\mathsf{k}_i := \mathsf{Decaps}_1(\mathsf{sk}_i, \mathsf{c})$
$\widetilde{\mathsf{k}} := \mathsf{Decaps}_0(\widetilde{\mathsf{sk}}, \widetilde{\mathsf{c}})$ //G_0,G_1
if $\exists \widetilde{\mathsf{k}}'$ s.t. $(\widetilde{\mathsf{pk}}, (\widetilde{\mathsf{c}}, \widetilde{\mathsf{k}}')) \in \widetilde{\mathcal{L}}$ //$\mathsf{G}_{0,0}$-$\mathsf{G}_{0,3}$
  $\widetilde{\mathsf{k}} := \widetilde{\mathsf{k}}'$ //$\mathsf{G}_{0,0}$-$\mathsf{G}_{0,3}$
else $\widetilde{\mathsf{k}} := \mathsf{Decaps}_0(\widetilde{\mathsf{sk}}, \widetilde{\mathsf{c}})$ //$\mathsf{G}_{0,0}$-$\mathsf{G}_{0,3}$
$\mathrm{ctxt} := (\mathsf{pk}_i, \mathsf{pk}_r, \widetilde{\mathsf{pk}}, \widetilde{\mathsf{c}}, \mathsf{c}_i, \mathsf{c}_r)$
$\pi'_r := \mathsf{G}_{\mathrm{R}}((\widetilde{\mathsf{k}}, \mathsf{k}_i, \mathsf{k}_r), \mathrm{ctxt})$
$\pi_i := \mathsf{G}_{\mathrm{I}}((\widetilde{\mathsf{k}}, \mathsf{k}_i, \mathsf{k}_r), \mathrm{ctxt})$
if $\pi'_r = \pi$
  $K := \mathsf{H}((\widetilde{\mathsf{k}}, \mathsf{k}_i, \mathsf{k}_r), \mathrm{ctxt})$
  $\mathtt{SK}[\mathrm{sID}] := K$
else
  $\mathtt{SK}[\mathrm{sID}] :=$ "reject"
  return $\bot$
$m_{i,2} := \pi_i$
return $m_{i,2}$

$\textsc{Test}(\mathrm{sID})$

if $\mathrm{sID} \in \mathcal{S}_{\mathtt{test}}$ **return** $\bot$ //already tested
if $\mathtt{SK}[\mathrm{sID}] = \bot$ **return** $\bot$
$\mathcal{S}_{\mathtt{test}} := \mathcal{S}_{\mathtt{test}} \cup \{\mathrm{sID}\}$
$K_0^* := \mathtt{SK}[\mathrm{sID}]$
if sID is type 1 or 2 in Table 2 //$\mathsf{G}_{0,2}$-$\mathsf{G}_{0,3}$,G_1
  $K_0^* \xleftarrow{\$} \mathcal{K}$
$K_1^* \xleftarrow{\$} \mathcal{K}$
return K_b^*

Fig. 25. Games sequence $\mathsf{G}_0, \mathsf{G}_{0,0}$-$\mathsf{G}_{0,3}, \mathsf{G}_1$ in proving Lemma 4. $\mathcal{A}$ has access to oracles $\mathrm{O} := \{\textsc{Session}_{\mathrm{I}}, \textsc{Session}_{\mathrm{R}}, \textsc{Der}_{\mathrm{I}}, \textsc{Der}_{\mathrm{R}}, \textsc{Reveal}, \textsc{Corr}, \textsc{Test}, \mathsf{G}_{\mathrm{R}}, \mathsf{G}_{\mathrm{I}}, \mathsf{H}\}$, where $\mathcal{A}$ has quantum access to random oracles $\mathsf{G}_{\mathrm{R}}, \mathsf{G}_{\mathrm{I}}$, and H. Omitted oracles are the same as in Figure 24.

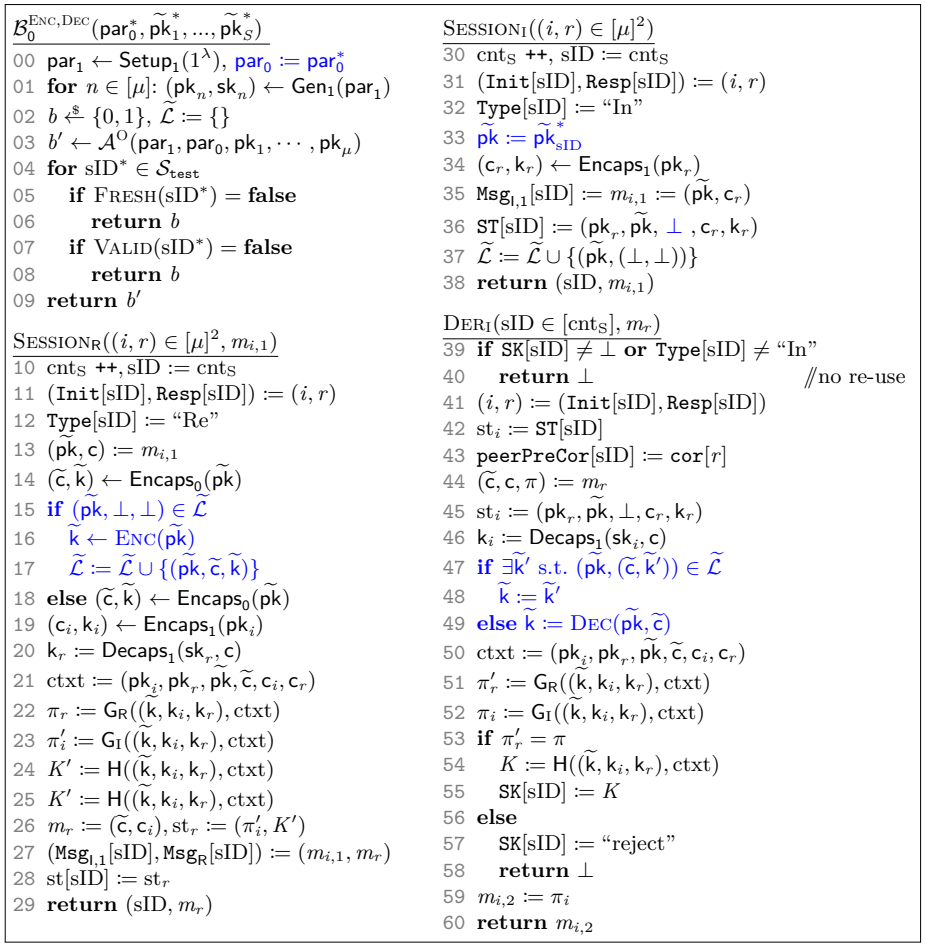

Fig. 26. Adversary $\mathcal{B}_0$ against MUC-IND-CCA. $\mathcal{B}_0$ simulates $\mathsf{G}_{0,0}$ or $\mathsf{G}_{0,1}$ for $\mathcal{A}$. $\mathcal{B}_0$ has access to oracles Enc and Dec. For sake of simplicity, here we only present the codes of $\textsc{Session}_{\mathrm{R}}$, $\textsc{Der}_{\mathrm{I}}$, and $\textsc{Der}_{\mathrm{I}}$, other oracles are the same as in $\mathsf{G}_{0,0}$. Highlight lines show how we embed the challenge from MUC-IND-CCA.

and the set of hash inputs of type-1 or type-2 tested sessions is viewed as the set $\mathcal{S}$ in Lemma 3. Now distinguishing $\mathsf{G}_{0,1}$ and $\mathsf{G}_{0,2}$ boils down to distinguishing QROs qRO_0 and qRO_1, and such distinguishing probability can be upper bounded by Lemma 3. Since $|\mathcal{S}|$ is bounded by the number of sessions in the AKE game and we view $\mathcal{S}$ as a uniformly random subset of $\mathcal{K}_0$, by Lemma 3, we have

$$\left|\Pr[\mathsf{G}_{0,1}^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathsf{G}_{0,2}^{\mathcal{A}} \Rightarrow 1]\right| \leq 2Q_{\mathsf{H}}\sqrt{S}/\sqrt{|\mathcal{K}_0|}.$$

Combining all the probability differences in the games sequence $\mathsf{G}_0, \mathsf{G}_{0,0}$-$\mathsf{G}_{0,3}, \mathsf{G}_1$, we have

$$\left|\Pr[\mathsf{G}_0^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathsf{G}_1^{\mathcal{A}} \Rightarrow 1]\right| \leq 2\varepsilon_0' + 2S\delta_0 + \frac{2Q_{\mathsf{H}}\sqrt{S}}{\sqrt{|\mathcal{K}_0|}}.$$

Proof (Lemma 5). We use intermediate games $\mathsf{G}_{2,0}$-$\mathsf{G}_{2,2}$ (cf. Figure 27) to bound the probability of $\Pr[\mathsf{ForgeKC}_{\mathrm{I}}]$. For simplicity, we say $\mathcal{A}$ triggers $\mathsf{ForgeKC}_{\mathrm{I}}$ if $\mathsf{ForgeKC}_{\mathrm{I}}$ is raised to **true**.

- Game $\mathsf{G}_{2,0}$. $\mathsf{G}_{2,0}$ proceeds identically with G_2 except that $\mathsf{G}_{2,0}$ outputs 1 if and only if $\mathcal{A}$ triggers $\mathsf{ForgeKC}_{\mathrm{I}}$ in G_2 (cf. lines 09 and 64). We have

$$\Pr[\mathsf{ForgeKC}_{\mathrm{I}} : \mathsf{G}_2^{\mathcal{A}}] = \Pr[\mathsf{G}_{2,0}^{\mathcal{A}} \Rightarrow 1].$$

- Game $\mathsf{G}_{2,1}$. We guess which initiator user will be impersonated by the adversary $\mathcal{A}$ when $\mathcal{A}$ causes $\mathsf{ForgeKC}_{\mathrm{I}}$ to be **true**, and if we guess wrong, then the game aborts with output 0. Concretely, we guess a user i^* uniformly at random before running $\mathcal{A}$ (cf. line 00), and abort the game with output 0 if $\mathcal{A}$

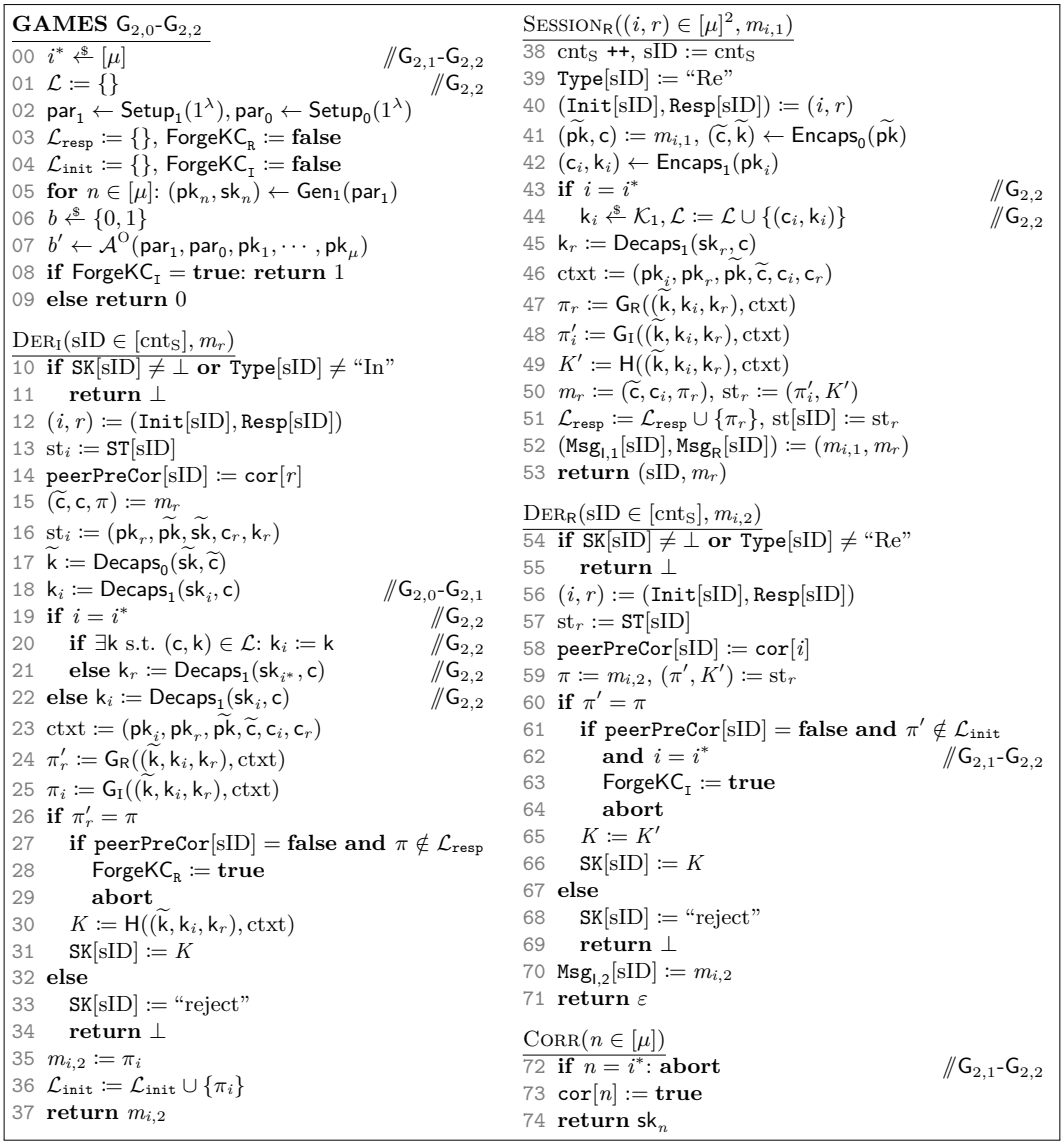

Fig. 27. Games sequence $\mathsf{G}_{2,0}$-$\mathsf{G}_{2,2}$ for bounding $\Pr[\mathsf{ForgeKC}_{\mathtt{I}}]$. Different to G_2, $\mathsf{G}_{2,0}$-$\mathsf{G}_{2,2}$ output 1 if and only if $\mathcal{A}$ causes $\mathsf{ForgeKC}_{\mathtt{I}}$ to be **true**. For simplicity, we only present the codes that different from G_2. All omitted oracles are the same as in G_2.

corrupts i^* (cf. line 72). $\mathsf{ForgeKC}_{\mathtt{I}}$ now is raised to **true** only if the initiator that $\mathcal{A}$ impersonates is i^* (cf. line 62). By a similar argument used in bounding $\Pr[\mathsf{G}^{\mathcal{A}}_{1,1} \Rightarrow 1]$, we have

$$\Pr[\mathsf{G}^{\mathcal{A}}_{2,1} \Rightarrow 1] = (1/\mu) \cdot \Pr[\mathsf{G}^{\mathcal{A}}_{2,0} \Rightarrow 1].$$

- Game $\mathsf{G}_{2,2}$. In $\textsc{Session}_{\mathrm{R}}$, if the initiator i is user i^*, then we sample k_i uniformly and independently at random instead of using Encaps_1 (cf. lines 43 to 44). We also use a list $\mathcal{L}$ (cf. line 01) that records all embedded challenge ciphertext-key pairs generated in $\textsc{Session}_{\mathrm{R}}$ (with initiator i^*) to make the simulation consistent (cf. line 01 and lines 19 to 22).

We construct a reduction $\mathcal{B}_1$ (in Figure 28) against the MC-IND-CCA security of KEM_1 to bound the difference between $\mathsf{G}_{2,1}$ and $\mathsf{G}_{2,2}$. $\mathcal{B}_1$ plays the MC-IND-CCA game with at most S challenge ciphertexts. It firstly samples i^* uniformly at random, initialize a list $\mathcal{L}$ to record all embed challenge ciphertext-key pairs, and set the public key of user i^* as the challenge public key. In $\textsc{Session}_{\mathrm{R}}$, if the initiator is i^*, $\mathcal{B}_1$ embeds a challenge ciphertext-key pair into the protocol messages and records it in $\mathcal{L}$. In $\textsc{Der}_{\mathrm{I}}$, if the initiator is i^*, $\mathcal{B}_1$ either uses the Dec oracle to decapsulate or finds the decapsulated key from $\mathcal{L}$.

$\mathcal{B}_1^{\text{Enc},\text{Dec}}(\mathsf{par}_1^*, \mathsf{pk}^*)$

00 $i^* \xleftarrow{\$} [\mu]$, $\mathcal{L} := \{\}$
01 $\mathsf{par}_1 := \mathsf{par}_1^*$, $\mathsf{pk}_{i^*} := \mathsf{pk}^*$
02 $\mathsf{par}_0 \leftarrow \mathsf{Setup}_0(1^\lambda)$
03 $\mathcal{L}_{\mathtt{resp}} := \{\}$, $\mathsf{ForgeKC}_{\mathtt{R}} :=$ **false**
04 **for** $n \in [\mu] \backslash i^*$:
05 $\quad(\mathsf{pk}_n, \mathsf{sk}_n) \leftarrow \mathsf{Gen}_1(\mathsf{par}_1)$
06 $b \xleftarrow{\$} \{0,1\}$
07 $b' \leftarrow \mathcal{A}^{\mathrm{O}}(\mathsf{par}_1, \mathsf{par}_0, \mathsf{pk}_1, \cdots, \mathsf{pk}_\mu)$
08 **return** 0

$\textsc{Der}_{\mathsf{R}}(\text{sID} \in [\text{cnt}_{\mathrm{S}}], m_{i,2})$

// This oracle is the same as in Figure 27
$\cdots$
09 **if** $\pi' = \pi$
10 $\quad$**if** $\mathtt{peerPreCor}[\text{sID}] =$ **false**
$\quad$**and** $\pi' \notin \mathcal{L}_{\mathtt{init}}$ **and** $i = i^*$
11 $\quad\quad\mathsf{ForgeKC}_{\mathrm{I}} :=$ **true**
12 $\quad\quad$**abort** and **return** 1
$\cdots$

$\textsc{Der}_{\mathrm{I}}(\text{sID} \in [\text{cnt}_{\mathrm{S}}], m_r)$

// Omitted parts are the same as in Figure 27
$\cdots$
13 $\widetilde{\mathsf{k}} := \mathsf{Decaps}_0(\widetilde{\mathsf{sk}}, \widetilde{\mathsf{c}})$
14 **if** $i = i^*$
15 $\quad$**if** $\exists \mathsf{k}$ s.t. $(\mathsf{c}, \mathsf{k}) \in \mathcal{L}$: $\mathsf{k}_i := \mathsf{k}$
16 $\quad$**else** $\mathsf{k}_i := \textsc{Dec}(\mathsf{c})$
17 **else** $\mathsf{k}_i := \mathsf{Decaps}_1(\mathsf{sk}_i, \mathsf{c})$
18 $\mathsf{ctxt} := (\mathsf{pk}_i, \mathsf{pk}_r, \widetilde{\mathsf{pk}}, \widetilde{\mathsf{c}}, \mathsf{c}_i, \mathsf{c}_r)$
$\cdots$

$\textsc{Session}_{\mathsf{R}}((i, r) \in [\mu]^2, m_{i,1})$

// Omitted parts are the same as in Figure 27
$\cdots$
19 $(\widetilde{\mathsf{c}}, \widetilde{\mathsf{k}}) \leftarrow \mathsf{Encaps}_0(\widetilde{\mathsf{pk}})$
20 $(\mathsf{c}_i, \mathsf{k}_i) \leftarrow \mathsf{Encaps}_1(\mathsf{pk}_i)$
21 **if** $i = i^*$
22 $\quad(\mathsf{c}_i, \mathsf{k}_i) \leftarrow \textsc{Enc}(), \mathcal{L} := \mathcal{L} \cup \{(\mathsf{c}_i, \mathsf{k}_i)\}$
23 $\mathsf{k}_r := \mathsf{Decaps}_1(\mathsf{sk}_r, \mathsf{c})$
24 $\mathsf{ctxt} := (\mathsf{pk}_i, \mathsf{pk}_r, \widetilde{\mathsf{pk}}, \widetilde{\mathsf{c}}, \mathsf{c}_i, \mathsf{c}_r)$
$\cdots$

Fig. 28. Adversary $\mathcal{B}_1$ against MC-IND-CCA. $\mathcal{B}_1$ simulates $\mathsf{G}_{2,1}$ or $\mathsf{G}_{2,2}$ for $\mathcal{A}$. $\mathcal{B}_1$ has access to oracles Enc and Dec. For sake of simplicity, here we only present the codes of $\textsc{Session}_{\mathsf{R}}$, $\textsc{Der}_{\mathrm{I}}$, and $\textsc{Der}_{\mathsf{R}}$, other oracles are the same as in $\mathsf{G}_{2,1}$. Highlight lines show how we embed the challenge from MC-IND-CCA.

If $\mathcal{B}_1$ plays $\mathsf{MC\text{-}IND\text{-}CCA}_{\mathsf{KEM}_1,0}$, then it perfectly simulates $\mathsf{G}_{2,1}$ for $\mathcal{A}$, and if it plays $\mathsf{MC\text{-}IND\text{-}CCA}_{\mathsf{KEM}_1,1}$, then it perfectly simulates $\mathsf{G}_{2,2}$. In line 15, we use the recorded KEM key without decapsulation, so we also need to consider the error bound δ_1 of KEM_1. Since there are S sessions in total, we have

$$\left|\Pr[\mathsf{G}_{2,1}^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathsf{G}_{2,2}^{\mathcal{A}} \Rightarrow 1]\right| \leq \varepsilon_1' + S\delta_1.$$

To makes $\mathsf{G}_{2,2}$ output 1, $\mathcal{A}$ needs to send a tag π to $\mathsf{Der}_{\mathsf{R}}$ such that (1) i^* is not corrupted yet when $\mathcal{A}$ sends π to $\mathsf{Der}_{\mathsf{R}}$, (2) π is not from $\textsc{Der}_{\mathrm{I}}$, namely, not recored in $\mathcal{L}_{\mathtt{init}}$, and (3) $\pi = \mathsf{G}_{\mathrm{I}}((\widetilde{\mathsf{k}}, \mathsf{k}_{i^*}, \mathsf{k}_r), \mathrm{ctxt})$ where k_{i^*} is uniformly random. By adjusting the argument used in bounding $\Pr[\mathsf{G}_{1,2}^{\mathcal{A}} \Rightarrow 1]$, we have

$$\Pr[\mathsf{G}_{2,2}^{\mathcal{A}} \Rightarrow 1] \leq 2Q_{\mathsf{H}}\sqrt{S}/\sqrt{|\mathcal{K}_0|} + S/2^\lambda.$$

By combining all probability differences, we have

$$\left|\Pr[\mathsf{G}_2^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathsf{G}_3^{\mathcal{A}} \Rightarrow 1]\right| \leq \Pr[\mathsf{ForgeKC}_{\mathrm{I}}] \leq \mu(\varepsilon_1' + S\delta_1 + \frac{2Q_{\mathsf{H}}\sqrt{S}}{\sqrt{|\mathcal{K}_0|}} + \frac{S}{2^\lambda}).$$

F Omitted Instiantiation: The CCGJJ Protocol

We can view the core of the protocol from Cohn-Gordon et al. [CCG+19] as a verifiable AKE protocol, ignoring the session key hash. We denote the underlying protocol by vCCGJJ (cf. Figure 29). Thus, our result describes an alternative approach to prove security of the extended protocol with key confirmation and an optimally tight proof which was recently shown in [GGJJ23]. We denote the protocol with key confirmation by $\mathsf{CCGJJ}_{\mathsf{KC}}$ which combines Figure 29 with Figure 5. In contrast to the proof in [GGJJ23] which loses the factor μ when adding key confirmation, we will lose it when proving that vCCGJJ is a OW-VwFS secure AKE protocol. As prior work we do not consider state reveals.

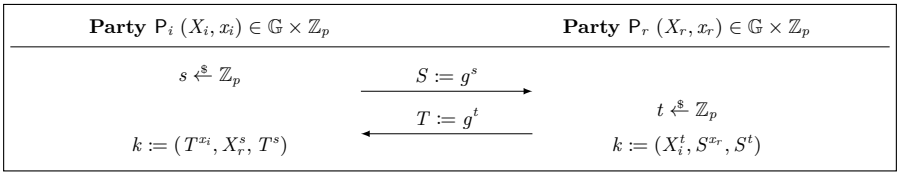

Fig. 29. AKE protocol vCCGJJ.

Security of vCCGJJ and $\mathsf{CCGJJ}_{\mathsf{KC}}$. Protocol vCCGJJ (and hence $\mathsf{CCGJJ}_{\mathsf{KC}}$) is perfectly correct and has public keys and messages with $\log(p)$ bits entropy. We prove security based on the StCDH assumption which asks to compute g^{xy} given (g^x, g^y) for $x, y, \xleftarrow{\$} \mathbb{Z}_p$ and access to a decision oracle $\mathrm{DH}_x(\cdot,\cdot)$. On input (Y, Z), DH_x returns a Boolean value whether $Y^x = Z$.

Lemma 6. *For every adversary $\mathcal{A}$ that breaks the $(t, \varepsilon, \mu, S, T, Q_{\mathrm{COR}}, Q_{Ver})$-OW-VwFS-security of vCCGJJ, there exist adversaries $\mathcal{B}_i$ for $i \in \{1,2,3\}$, that break $(t_i, \varepsilon_i, Q_{DDH,i})$-StCDH with $t_i \approx t$ and $\varepsilon \leq \varepsilon_1 + \mu \cdot (\varepsilon_2 + \varepsilon_3)$, where $Q_{DDH,i} \leq Q_{Ver} + 1$ for all $i \in \{1,2,3\}$.*

The proof is very similar to that of the original CCGJJ protocol. From this, we can recover the result for the $\mathsf{CCGJJ}_{\mathsf{KC}}$ protocol provided by [GGJJ23].

Theorem 7. *Let $\mathsf{CCGJJ}_{\mathsf{KC}}$ be the protocol obtained by adding key confirmation as described in Figure 5 to protocol vCCGJJ. Then for every adversary $\mathcal{A}$ that breaks the $(t, \varepsilon, \mu, S, T, Q_{\mathrm{COR}})$-IND-FS-security of $\mathsf{CCGJJ}_{\mathsf{KC}}$, there exists an adversary $\mathcal{B}$ that breaks $(t', \varepsilon', Q_{DDH})$-StCDH with $t' \approx t$ and*

$$\varepsilon \leq \Theta(\mu) \cdot \varepsilon' + (S + S^2) \cdot 2^{-\lambda} + (\mu^2 + S^2 + Q_{\mathsf{G}_\mathsf{I}} + Q_{\mathsf{G}_\mathsf{R}} + Q_{\mathsf{H}}) \cdot 2^{-\log(p)} \ ,$$

where $Q_{\mathsf{G}_\mathsf{I}}, Q_{\mathsf{G}_\mathsf{R}}$ and Q_{H} are the number of queries to random oracles G_I, G_R and H, $Q_{DDH} \leq S + Q_{\mathsf{G}_\mathsf{I}} + Q_{\mathsf{G}_\mathsf{R}} + Q_{\mathsf{H}} + 1$.

Proof (Theorem 7). The theorem follows from combining Lemma 6 and Theorem 1, where we fold adversaries $\mathcal{B}_1$, $\mathcal{B}_2$ and $\mathcal{B}_3$ into a single adversary. To bound the advantage from Lemma 6, we use a similar argument as described in Remark 2.

Proof (Lemma 6). Let $\mathcal{A}$ be an adversary that breaks the $(t, \varepsilon, \mu, S, T, Q_{\mathrm{COR}}, Q_{Ver})$-OW-VwFS-security of vCCGJJ. That is $\mathcal{A}$ computes the key $k^* = (k_1^*, k_2^*, k_3^*)$ for a fresh and valid sID^* according to game OW-VwFS.

We proceed similar to the proof of Lemma 1, i. e., we will make a case distinction, only this time, we will consider case (2) and (3) separately and construct three adversaries in total. We will bound ϵ by

$$\epsilon \leq \sum_{i \in \{1,2,3\}} \Pr[\mathsf{OW\text{-}VwFS}^{\mathcal{A}}_{\mathsf{vCCGJJ}} \Rightarrow 1 \wedge \text{Case } (i)] \ .$$

Case (1). We start with adversary $\mathcal{B}_1$ which is given in Figure 30. $\mathcal{B}_1$ gets as input a CDH challenge $(\bar{S}, \bar{T})$ and has access to a decision oracle $\mathrm{DH}_{\bar{S}}$. We will embed $\bar{S}$ in initiator sessions and $\bar{T}$ in responder sessions, re-randomizing group elements. The idea is that when $\mathcal{A}$ computes a session key for a matching session (note that it may have corrupted both parties in this case), k_3 will give us the CDH solution.

Fig. 30. Adversary $\mathcal{B}_1$ against StCDH for the proof of Lemma 6. $\mathcal{A}$ has access to oracles $\mathrm{O} = \{\textsc{Session}'_I, \textsc{Der}'_R, \textsc{Der}'_I, \textsc{Corr}', \textsc{KVer}\}$. Procedure FindMatch is a helper algorithm, not available to the adversary. Highlight lines show how we embed the StCDH challenge.

We now describe $\mathcal{B}_1$ in more detail. It first generates all long-term keys which will allow it to answer all queries to $\textsc{Corr}'$. For queries to $\textsc{Session}'_I$, $\mathcal{B}_1$ picks s to randomize $\bar{S}$ and stores s in the state variable. For queries to $\textsc{Der}'_R$, $\mathcal{B}_1$ will only embed the challenge when there is a partially matching session. In this case it randomizes $\bar{T}$ and computes k_1 and k_2 using the long-term keys of the respective parties. It marks k_3 with a special symbol $\diamond$ and stores randomization exponent t in an additional state variable. If there is no partially matching session than $\mathcal{B}_1$ simply follows the protocol which allows to compute the full session key. On a query to $\textsc{Der}'_I$, $\mathcal{B}_1$ also computes the session key as $(k_1, k_2, \diamond)$ because it does not know the actual exponent for k_3.

Queries to KVer can be simulated as follows: If the session ID sID provided by $\mathcal{A}$ has stored $k_3 = \diamond$ for correct values of k_1 and k_2, we have to consider two cases. First, there exists a (partially) matching session to sID which we can find via the helper procedure FindMatch, we retrieve the exponents used for randomization and check whether k_3 is valid using the $\mathrm{DH}_{\bar{s}}$ oracle. If it returns **true**, then $\mathcal{B}_1$ stops immediately and outputs the correct solution. Second, if there is no (partial) matching session, then we know this is an initiator session and use exponent s to de-randomize k_3 and check its validity using the $\mathrm{DH}_{\bar{s}}$ oracle. In all other cases, we know the complete session key and can simulate KVer consistently.

Finally, if $\mathcal{B}_1$ has not yet stopped, $\mathcal{A}$ returns $(\mathrm{sID}^*, (k_1^*, k_2^*, k_3^*))$ and $\mathcal{B}_1$ computes the checks of KVer. If sID^* has a (partially) matching session and $\mathcal{A}$ has computed the correct key, $\mathcal{B}_1$ outputs the correct CDH solution. We have $\Pr[\mathsf{OW\text{-}VwFS}^{\mathcal{A}}_{\mathsf{vCCGJJ}} \Rightarrow 1 \wedge \text{Case (1)}] \leq \varepsilon_{\mathcal{B}_1}$, which analysis the description of $\mathcal{B}_1$.

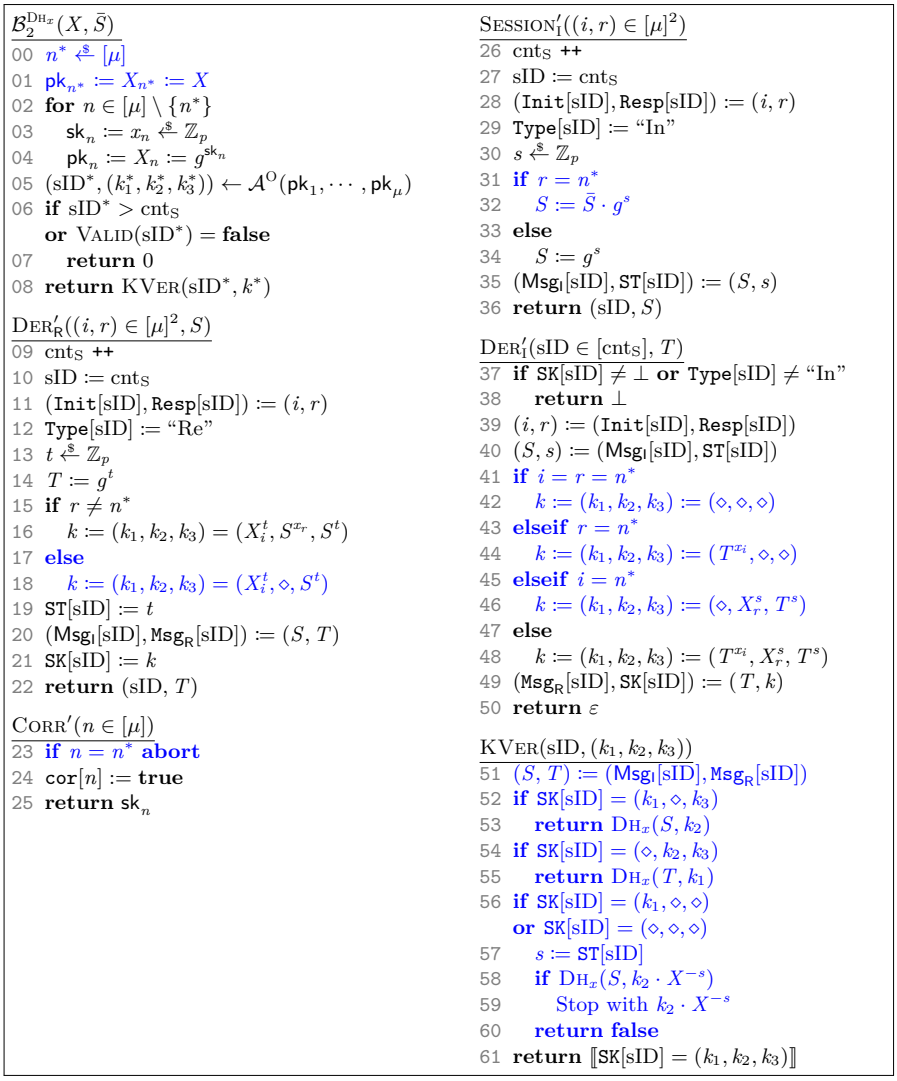

Fig. 31. Adversary $\mathcal{B}_2$ against StCDH for the proof of Lemma 6. Highlight lines show how we embed the StCDH challenge.

Now we turn to the case, where the session that $\mathcal{A}$ outputs does not have a (partially) matching session. We proceed similar to the proof of the full CCGJJ protocol in [CCG+19] and consider initiator and responder sessions separately.

Case (2). We will start with initiator sessions and construct adversary $\mathcal{B}_2$ in Figure 31. $\mathcal{B}_2$ gets as input a CDH challenge $(X, \bar{S})$ and has access to a decision oracle DH_x. It first guesses a user n^* and hopes that this user will be the responder of the session sID^* of type "In". Recall that in this case the responder of sID^* must not be corrupted. For all other users, it chooses long-term key pairs itself. If $\mathcal{A}$ wants to corrupt user n^*, then $\mathcal{B}_2$ aborts.

The other oracles are simulated as follows. The challenge $\bar{S}$ will be embedded in all queries to $\textsc{Session}'_I$, where n^* is the intended peer. We randomize $\bar{S}$ as in the previous case. In all other sessions, $\mathcal{B}_2$ computes the message as in the protocol. We only care about initiator session which do not have a partner, so queries to $\textsc{Der}'_R$ are answered as in the protocol. However, if the holder of the session is n^*, we do not know how to compute k_2 and put a placeholder symbol $\diamond$. When $\textsc{Der}'_I$ is queried, then we need to do another case distinction: (1) if the holder and peer of the session are the same party n^*[9], then we may not know how to compute any k_i and put placeholders everywhere. If the peer is n^*, then we also embedded $\bar{S}$ and we will not be able to compute k_2 and k_3. If the holder of the session is n^*, then we will not be able to compute k_1. In all other cases, we can simply compute the key.

[9] This captures reflection attacks.

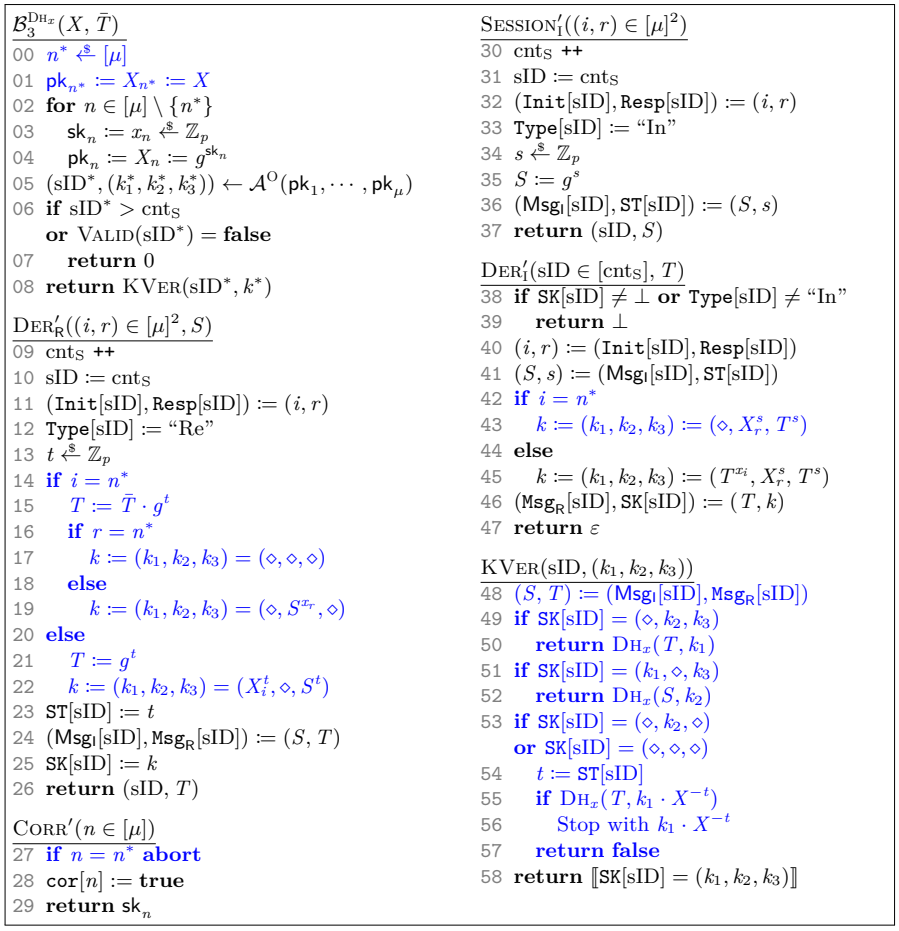

Fig. 32. Adversary $\mathcal{B}_3$ against StCDH for the proof of Lemma 6. Highlight lines show how we embed the StCDH challenge.

Based on this, we will now describe how to simulate the KVER oracle consistently. If only k_2 is unknown, we know this is a responder session and we can use DH_x to decide whether this is the correct session key. If only k_1 is unknown, then this is an initiator session where n^* is the holder (and thus not relevant for this case), so we also use DH_x to check whether the key is correct. If k_2 and k_3 or all key values are unknown, then this is a session where n^* is the peer and we will use k_2 to either solve the CDH challenge or decide that the key is incorrect. We do this by removing the randomization s from k_2. Then we have $k_2 \cdot X^s = g^{x(\bar{s}+s)} \cdot g^{xs} = g^{x\bar{s}}$ and can stop immediately. If DH_x outputs **false**, then we know the session key cannot be correct and $\mathcal{B}_2$ outputs **false** as well.

Finally, if $\mathcal{B}_2$ has not yet stopped (or aborted), $\mathcal{A}$ returns $(\mathrm{sID}^*, (k_1^*, k_2^*, k_3^*))$ and $\mathcal{B}_2$ computes the checks of KVER. $\mathcal{B}_2$ will be successful if n^* is indeed the peer of session sID^* and sID^* is of type "In". Thus, we have $\varepsilon_{\mathcal{B}_2} \geq \frac{1}{\mu} \cdot \Pr[\mathsf{OW\text{-}VwFS}^{\mathcal{A}}_{\mathsf{vCCGJJ}} \Rightarrow 1 \wedge \text{Case (2)}]$, which analysis the description of $\mathcal{B}_2$.

CASE (3). Finally, we look at responder sessions where the peer is not corrupted. We construct an adversary $\mathcal{B}_3$ in Figure 32 which gets as input a CDH challenge $(X, \bar{T})$ and access to oracle DH_x. $\mathcal{B}_3$ works exactly as adversary $\mathcal{B}_2$, except that it embeds the challenge in responder sessions. More specifically, it simulates $\textsc{Session}'_I$ as in the protocol. Then it embeds $\bar{T}$ whenever the peer of a query to $\textsc{Der}'_R$ is n^* and computes all keys it can trivially compute, marking all others with $\diamond$. Keys for queries to $\textsc{Der}'_I$ can be computed except if the holder of the session is n^*. Oracle KVER is also exactly simulated as adversary $\mathcal{B}_2$ did.

We get $\varepsilon_{\mathcal{B}_3} \geq \frac{1}{\mu} \cdot \Pr[\mathsf{OW\text{-}VwFS}^{\mathcal{A}}_{\mathsf{vCCGJJ}} \Rightarrow 1 \wedge \text{Case (3)}]$, which concludes the proof of Lemma 6.